SPECTRUM GRAPHICS AND SOUND

STEVE MONEY

Spectrum Graphics and Sound

Other Granada books for ZX Spectrum users

THE ZX SPECTRUM
And How To Get The Most From It
Ian Sinclair
0 246 12018 5

THE SPECTRUM PROGRAMMER
S. M. Gee
0 246 12025 8

THE SPECTRUM BOOK OF GAMES
M. James, S. M. Gee and K. Ewbank
0 246 12047 9

INTRODUCING SPECTRUM MACHINE CODE
Ian Sinclair
0 246 12082 7

Spectrum Graphics and Sound

Steve Money

GRANADA
London Toronto Sydney New York

Granada Technical Books
Granada Publishing Ltd
8 Grafton Street, London W1X 3LA

First published in Great Britain by
Granada Publishing 1984

British Library Cataloguing in Publication Data
Money, Steve A
Spectrum graphics and sound
1. Computer graphics 2. Sinclair ZX
Spectrum (Computer)
I.Title
001.64'43 1385

ISBN 0 246 12192 0

Typeset by V & M Graphics Ltd, Aylesbury, Bucks
Printed and bound in Great Britain by
Mackays of Chatham, Kent

Contents

Preface

One of the most popular home computers with a colour display capability is the Spectrum. With its high resolution graphics capability the Spectrum allows quite good drawings to be produced on the display screen. This can be very useful for displaying graphs and charts and of course is an essential ingredient of the many games programs that have been written for the Spectrum. In this book we shall explore the techniques of producing drawings and charts on the screen.

The first chapter explains in simple terms how the display on the screen is created and looks at some of the novel features used in the Spectrum.

In Chapter Two we move on to explore the possibilities for using the mosaic graphics symbols which are included in the standard symbol set of the Spectrum. The symbols are printed on to the screen in the same way as text characters. The Spectrum, unlike some other computers, does not provide plotting or drawing commands for use with the mosaic graphics but in Chapter Two we shall explore ways of handling the mosaic symbols to draw lines and even to provide a simple sketchpad program.

Perhaps the most attractive feature of the Spectrum is its high resolution colour graphics capability. In Chapters Three and Four we explore the techniques involved in drawing various kinds of figure such as the polygon and circle. Unfortunately the method used by the Spectrum for storing its colour information imposes some limitations on the use of colour in the high resolution mode but nevertheless very good results can be obtained with a little care.

The Spectrum does provide facilities for producing custom designed symbols and some of the possibilities of this feature are examined in Chapter Five. It is quite easy to produce very large versions of the symbols on the screen if desired and programs are given which permit this to be done.

It is possible to produce rather more shades of colour than the eight basic colours provided by the INK and PAPER commands of the Spectrum and in Chapter Six the techniques for doing this are explored.

One of the more practical applications of the Spectrum is to display graphs and charts of various kinds and some of the techniques involved in drawing graphs, bar charts and pie charts are shown in Chapter Seven.

For games, of course, movement is an important ingredient and in Chapter Eight we explore the basic principles and techniques that can be used to animate objects on the display screen. Unfortunately the BASIC language of the Spectrum is relatively slow and for really high speed action it is generally necessary to resort to writing programs directly in machine code which is a topic beyond the scope of this book.

It is possible to produce pseudo three-dimensional images on the Spectrum display. This can be done by drawing three axis charts and graphs or by using perspective techniques to provide more realistic views of scenes or objects. A program is included which permits an object to be viewed from any angle.

Finally we come to the generation of sound and music. In this respect the Spectrum is rather limited since it has only one simple BEEP command for producing sound. Nevertheless the Spectrum can be made to play tunes and can also be converted into a simple musical instrument as we shll see in Chapter Ten.

The aim of this book has been to explain some of the techniques for using the graphics and sound facilities of the Spectrum. Much of the fun of playing with computers however comes from exploring new ideas and it is hoped that this book will at least provide a guide to allow you to explore the possibilities of the Spectrum computer.

Steve Money

Chapter One

Introducing Graphics

An attractive feature of almost all of the modern personal or home computers is their ability to produce colourful graphics displays. This facility permits the computer to be programmed so that it will present drawings, pictures, graphs, charts and animated displays on the TV screen.

All of the popular home computer systems can produce graphics of some sort and most of them have the added attraction of colour. The quality of the displays that are produced depends upon what is known as the graphics *resolution*, which is a measure of how fine the details can be in a displayed picture. Resolution is measured as the number of points across and down the screen which can be individually controlled. Thus a resolution of 256 × 176 means there are 256 points across the screen and 176 points down the screen, each of which can be set or reset. In general the higher the resolution, the better the displayed picture on the screen.

One problem with high resolution graphics displays is that they tend to require large amounts of memory. When the display is in colour even more memory is needed to store information about the colour of the individual dots on the screen. To save memory some computers limit the number of colours that can be displayed at a time. In the Spectrum a display technique is used which does, with some limitations, allow the use of eight different colours simultaneously without tying up too much of the available memory.

One of the more popular uses for home computers is to emulate the video games normally found in an amusement arcade. Typical of these games are Invaders, Asteroids, Defender and Pacman. The popular arcade games rely heavily on graphics displays with brilliant colours and fast moving action on the screen. Actual arcade games use microprocessors similar to those used in home computers. In fact the games machines are often quite powerful computers in their own right, but are dedicated to the task of providing a specific video game.

Arcade games can be presented on a home television by using a dedicated video games console, such as the type made by Atari. These units also use a microprocessor but unlike the arcade machine they can generate a wide selection of games. In these machines the computer program for each different game is supplied in a cartridge which is plugged into the games console unit.

Computers as games machines

A home computer, such as the Spectrum, can also be turned into an arcade games machine provided that it has a good graphics display capability. Here the computer is programmed to display a real time moving picture of the game situation with perhaps rockets, spaceships and aliens moving rapidly around the TV screen. Colour and sound are also important constituents for such games and in fact the Spectrum can perform quite well on all of these points.

A major advantage of the home computer compared with a video game unit is that you can invent your own arcade games and program them into the computer using either BASIC or machine code. Having written the program it can then be saved on a cassette tape for future use.

Apart from arcade type games here are many other varieties of game that can be played on a computer. These range from simple number guessing games up to complex games such as chess. Of these games one of the more popular types is the adventure game. Variations on this theme include maze games and the role playing games like Dungeons and Dragons. In adventure games the player explores a mysterious world, moving from location to location seeking treasures, fighting monsters and solving riddles. The computer itself acts as your puppet telling you what it can see and carrying out actions such as picking up objects. The Dungeons and Dragons versions allow the player to adopt the role of various characters as he explores.

In most adventure type games the locations are described by written text on the screen and these are generally referred to as 'text' adventures. Adventure games can be greatly enhanced by actually showing on the TV screen a picture of the current location of the explorer rather than just having a written description. It is also possible to show pictures of various objects or treasures that may be in the explorer's current location. In some versions, the explorer may be allowed to examine a selected object in detail. Here the image of the object is expanded to fill the screen.

A simpler variant of the adventure game is the maze. Here the explorer is dropped into the middle of a maze and has to find his way to the exit. Simple versions describe the possible directions of travel in words but a more attractive version shows a picture of what the explorer can see when he looks along the path he is on. By drawing a perspective view looking along the corridor in the maze the side turnings can be shown in fairly realistic form. As the player moves the picture on the screen is updated to show the view from the new position. The ultimate in this type of game is a three dimensional maze where the explorer may be able to move not only forward, backward, left and right, but also up and down through various levels of the maze.

Other uses for graphics

Games are, of course, a pleasant pastime but most computer owners and users eventually use the computer for more serious activities as well. For many applications a chart or graph can be much more effective than a list of numbers in showing how a business or any other activity is progressing. Graphs and charts are also important in scientific applications where we want to display the results of a scientific experiment or the information gained from a poll or survey. The various types of display of results, such as bar charts, pie charts and conventional graph plots, can all be produced quite effectively on the Spectrum.

Drawing on a computer using a high resolution graphics display is another application. We might perhaps produce technical drawings such as plans, electronic circuit diagrams, or book illustrations. Many modern drawing offices use computer aided techniques for drawing and designing which are provided by a set of programs called a *computer aided design* (CAD) package. We might use similar techniques to show the apparatus and demonstrate the results of a physics or chemistry experiment. This could be very useful where the actual equipment required would be very expensive. Here the computer simulates the actual experiment and the display might show the various stages in the progress of the experiment giving readouts of perhaps temperature, pressure, etc.

Computer graphics can of course be used purely as an art form. Here the computer will be producing pretty patterns, handling colour and perhaps producing perspective views. Using techniques for displaying solid objects it becomes possible to draw an object and then view it from different directions. The shape can then perhaps be modified and the result viewed until the desired effect is obtained.

The video display

Modern home computers, including the Spectrum, use a television screen to provide a display of output from the computer program. This may use the domestic television receiver or alternatively a special television monitor designed for use as a computer visual display unit (VDU). The monitor type display is particularly useful for colour displays and will give clearer and steadier pictures since the electronic techniques normally used in television broadcasting and reception do tend to degrade the graphics picture. At this stage it might be useful to take a look at the way in which the actual display is produced on the screen.

Although the image on the television screen looks like a steady complete picture it is actually traced out by a single moving dot which sweeps very rapidly across the screen in a series of horizontal lines. As the dot moves across the screen it also moves rather more slowly down the screen so that as each successive line is produced it falls just below the last one traced out on the screen. When the bottom right corner of the screen is reached the dot moves quickly back up to the left-hand top corner and the whole scanning process is repeated again. The scanning operation goes on continuously with the dot sweeping down the screen every fiftieth of a second. The total number of scan lines is 625.

When we view this display our eyes are unable to see the dot moving because it travels so fast and the eye does not respond to things that change faster than about 20 times a second. The screen material is also designed so that as the dot moves along, the points which were lit do not switch off immediately but fade out in about $1/20$ of a second. As a result we see the whole picture as if it were present on the screen continuously. To reduce flickering the scan lines are interlaced. What happens is that on one sweep down the screen the dot traces out every other line, say the odd numbered lines, then on the next sweep it fills in the gaps by tracing out the even numbered lines. This gives a flicker rate of 50 per second which the eye cannot see, whereas tracing all of the lines once every $1/_{25}$ second could produce a noticeable flicker which might be distracting to the viewer.

As the spot sweeps over the screen its brightness can be varied so that a pattern of light and dark dots appears and these make up the picture that we see. A colour television display has a special screen with dots giving red, green or blue light, arranged in tiny triangular groups. Three separate scanning beams operate inside the tube which select red, green and blue dots respectively. By using combinations of red, green and blue light from each set of three dots any point on the screen can be set to any desired colour.

Text displays

When you switch on your Spectrum computer it will display a white screen with black text written on it. Before going on to look at graphics displays it might be useful if we take a look at the way in which a typical home computer generates the display of printed text symbols on the TV screen.

For displaying text the screen is effectively divided up into an array of small rectangular spaces, each of which is called a symbol, or character, space. In each of these symbol spaces on the screen a single letter, number or other symbol can be displayed. When presenting text the Spectrum computer divides its screen up into a total of 24 horizontal rows with 32 symbols across each row. As we shall see later two rows of the display are reserved for use by the computer, leaving 22 rows for the normal text display.

If the text symbols on the screen are examined closely it will be seen that each letter is made up from a small array of individual dots. The Spectrum uses an array consisting of eight rows of eight dots each within the character space. Now the shape of the text symbol can be produced by lighting up some of the dots and leaving the others dark. An example of this is shown in Fig. 1.1. By selecting the required patterns to be displayed in each symbol space the complete picture can be built up on the screen. The Spectrum prints its text as black letters and these normally appear against a white background. In this case, the dots picking out the shape of the letter are turned off.

Since the complete picture is traced out fifty times a second we need to have some data, representing all of the text to be displayed, held somewhere in memory so that the display logic can call up the

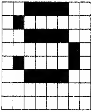


Fig 1.1 The dot pattern matrix used to display text symbols

dot patterns as it needs them to produce the screen display The usual arrangment of the display system is shown in simplified form in Fig 1.2 A section of memory, usually called the display memory, is used to hold data which defines the text symbols that are to be displayed. Typically an 8 bit data code is allocated to each of the symbols that can be displayed Each memory location also contains 8 data bits so that each text symbol takes up one location in the display memory

The dot patterns making up the different symbols are held in a special memory device called a *character generator*. This is basically a memory device but unlike the normal computer memory the data patterns are permanently written into it when it is made and they cannot be erased or changed. Such a memory is called a Read Only

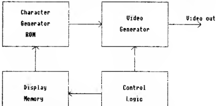


Fig 1.2 Basic system arrangement for a computer text display

Memory or ROM. When the character code is applied to the character generator it selects the appropriate pattern of dots and the data representing the dots can then be read out to generate the video signals that will produce the actual dots on the screen.

As the picture is traced out the data for each character space is read from the display memory and this is used to call up the required dot pattern from the character generator. The dot patterns are then converted into a video signal similar to that which produces our familiar television pictures, and when this is fed to a television receiver or monitor the text display is produced on the screen. The actual electronic logic required is quite complex and in most computers a special dedicated integrated circuit or 'chip' is used to control the generation of the screen display.

The Spectrum is rather different from most other computers in the way it produces its text displays. Like the other machines it does have a section of memory which permanently contains the dot patterns for the set of symbols that can be displayed. However, instead of storing the code for a text symbol in its display memory, the Spectrum stores the actual dot pattern. Thus when a character is called up for display its dot pattern is copied from the character generator into the display memory. Because each symbol pattern contains eight rows of dots the screen display memory is much larger that one where only the character code is stored. In fact the memory used in the Spectrum is eight times larger than it would be using conventional techniques. As we shall see however this technique has advantages, especially where high resolution graphics pictures and text symbols are to be mixed on the display screen.

Mosaic graphics

In the early days of computers, graphs and pictures were often produced by actually using text symbols to make up individual points in the picture. Some letters will appear brighter than others because of their shape, so by carefully choosing the letter placed at a point, the image may be made light or dark. If the resultant page of text is viewed from a sufficient distance then it will show a picture, since the viewer's eyes will not be able to resolve the individual letters. This technique does not work particularly well on the Spectrum because there are not enough symbols on the screen to give a usable picture.

Using the normal text symbols to produce pictures is not particularly effective and many home computers have additional special symbols in their character set to allow pictures to be drawn on the screen. These special symbols might contain a segment of a line running vertically, horizontally or diagonally through the character space. Other symbols might display corner or T-junction shapes or even curved lines. By carefully placing these special symbols on the screen quite detailed drawings and pictures can be produced. Other special symbols available might include such things as playing card symbols or even pictures of chess pieces. Some computers allow the user to create custom symbols by storing the dot patterns in the main computer memory rather than in a special character generator ROM.

This technique of using special graphics characters to build a picture can be rather restricted unless a very wide selection of special symbols is used. The Spectrum does not provide special graphics symbols as standard but there is a facility by which the user can create special symbols to his own design.

There is an alternative and more flexible approach to the production of low to medium resolution graphics displays. The Spectrum makes use of what are known as *mosaic* graphics symbols similar to those used for producing graphics displays on the teletext and viewdata services. Once again an extra set of special graphics symbols is used, but in this case the character space is divided up into a pattern of four small blocks. Each of the block elements may be lit or dark to form a coarse pattern within the character space. Figure 1.3 shows typical examples of mosaic graphics symbols. In much the same way as the special line graphics symbols were used, the block patterns of the mosaic symbols can be built up to form a picture this technique gives a rather coarser looking picture than the use of special line segment symbols but it is more flexible.

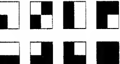


Fig. 1.3 Typical mosaic graphics symbols as used on the Spectrum.

In the Spectrum each symbol space is divided up into four smaller blocks. If each block can be either black or white there are sixteen different patterns of blocks that can be produced within the text symbol space. By carefully choosing the block patterns within the text spaces it is possible to produce a picture on the screen. Here we have a rather crude form of television picture. On the Spectrum machine this type of graphics display provides a resolution of 64 dots across the screen and 44 down the screen.

Although the Spectrum normally produces black text symbols it is possible to display the text and mosaic graphics symbols in eight different colours and it is also possible to set the background colour of each symbol space to any one of the eight colours. Thus we are able to produce quite colourful pictures using this form of graphics display. To get some idea of the resolution and colour capability of the mosaic graphics display try running the program listed in Fig 1.4.

```
100 REM Mosaic graphics demo
110 LET x=15
120 LET y=10
130 FOR j=1 TO 15
140 INK INT (RND*7)
150 LET k=j
160 IF k>10 THEN  LET k=10
170 FOR i=1 TO j
180 PRINT AT y+k,x+i;CHR$ 130;
190 PRINT AT y+k,x-i;CHR$ 129;
200 PRINT AT y-k,x-i;CHR$ 132;
210 PRINT AT y-k,x+i;CHR$ 136;
220 NEXT i
230 FOR i=1 TO k
240 PRINT AT y+i,x+j;CHR$ 130;
250 PRINT AT y+i,x-j;CHR$ 129;
260 PRINT AT y-i,x-j;CHR$ 132;
270 PRINT AT y-i,x+j;CHR$ 136;
280 NEXT i
290 NEXT j
300 FOR n=1 TO 500
310 LET i=INT (RND*16)
320 LET j=INT (RND*10)
330 INK INT (RND*7)
340 PRINT AT y+j,x+i;CHR$ 130;
350 PRINT AT y+j,x-i;CHR$ 129;
360 PRINT AT y-j,x-i;CHR$ 132;
370 PRINT AT y-j,x+i;CHR$ 136;
380 NEXT n
```

Fig 1.4 Program demonstrating the mosaic graphics on the Spectrum

Using mosaic symbol graphics to draw lines, or even to place a point at some specific position, can become quite complex as we shall see in the next chapter. The Spectrum can be programmed so that individual mosaic blocks can be selected and either lit or turned off to produce a pattern or even a picture. There are however some limitations in the use of colour when this technique is used but nevertheless some quite attractive displays can be produced.

One advantage of the mosaic symbol graphics is that since they can be printed on to the screen in the same way as text symbols, it is easy to mix text and graphics on the display.

Special characters and graphics

A technique used on some computers for producing graphics is to have a special set of graphics symbols such as horizontal, vertical and diagonal lines. There might also be curved lines and even special symbols such as the suit signs for playing cards. By carefully selecting and placing these special symbols on the screen better looking graphics than the mosaic type can be produced.

The Spectrum has a facility for producing such custom or do-it-yourself symbols. For these special symbols the dot patterns can be set up in a reserved area of the main computer memory and will be retained there whilst the computer remains switched on. Unlike the normal text symbols, however, the patterns will be lost when the computer is turned off. The dot patterns are transferred to the main screen memory in the same way as those of the normal text and mosaic graphics symbols.

Sometimes a larger graphics pattern, with perhaps a dot array of 16×16 dots may be required and can be built up from a group of standard size symbols.

High resolution pixel graphics

As we have already seen, the text symbols on the screen are themselves produced by selectively lighting dots within each symbol space on the screen. The available dot patterns are, however, fixed since they are governed by the sets of dot patterns held in the character generator.

Suppose we had an alternative display mode where we could control the states of the individual dots in every character space on the screen. In the Spectrum each text symbol is made up from 8 rows of dots in each row. With 32 character spaces across the screen this gives 32 × 8 or a total of 256 dots across the screen for the horizontal or X resolution. Although there are in fact 24 displayed rows of text on the screen, the bottom two rows are reserved for use by the computer when it displays messages so that only 22 rows are available for use as a display. Since the character space is 8 dots high and there are 22 rows of text there will be 8 × 22 or a total of 176 dots down the screen.

```
100 REM Hi-res graphics demo
110 LET x=0
120 LET y=0
125 REM Draw vertical lines
130 FOR w=1 TO 22
140 PLOT x,y
150 DRAW 0,175
160 LET x=x+w
170 NEXT w
180 LET x=0
185 REM Draw horizontal lines
190 FOR h=1 TO 19
200 PLOT x,y
210 DRAW 255,0
220 LET y=y+h
230 NEXT h
235 REM Draw diagonal lines
240 PLOT 0,0
250 DRAW 255,175
260 PLOT 0,175
270 DRAW 255,-175
275 REM Draw border
280 PLOT 0,0
290 DRAW 255,0
300 DRAW 0,175
310 DRAW -255,0
320 DRAW 0,-175
```

Fig. 1.5 Program demonstrating high resolution graphics on the Spectrum.

The individual dots in each character space are generally referred to as *pixels* (picture elements) and this mode of graphics is called pixel graphics or *high resolution* graphics. Figure 1.6 gives an example of the type of display that can be produced using the high resolution graphics capability of the Spectrum. In fact many of the illustrations in this book were actually generated by the Spectrum and then printed out on a dot matrix printer

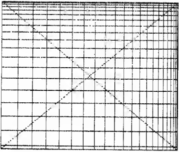


Fig 1 6 Screen display produced by the program of Fig 1 5

In the following chapters of the book we shall explore the facilities of both low and high resolution graphics and discover the techniques of drawing shapes, painting in colour, creating new symbols and producing graphs or charts. We shall also look at the principles of animation and the creation of perspective and pseudo three-dimensional displays.

Chapter Two

Low Resolution Graphics

Let us start on our exploration of Spectrum graphics by looking at the *low resolution* graphics which effectively use the text display screen

Using text symbol spaces

In the early days of computers all output from the computer was printed out as text symbols on a printer and there were usually no facilities for providing graphics displays. Programmers overcame this limitation by using text symbols as the individual picture elements to build up a graphics picture Symbols such as M or W would produce a dark grey shade whilst a full stop gives a very light coloured picture element By carefully choosing the type and position of the text symbols a picture could be built up on the paper which would look quite good, provided that it was viewed from a sufficient distance to ensure that the text symbols could not be picked out individually Using this technique quite respectable pictures of animals or the faces of well known personalities can be produced. This technique doesn't work very well on the Spectrum display screen but could be used if an 80 column printer were interfaced to the computer and a printout of perhaps 100 lines of text were used to build up the picture This would give some 8000 individual picture elements in an area of about 8 × 10 in on the paper and, if viewed from about eight feet, the resultant pictures can be quite impressive. A similar technique has been used to print pictures on tee shirts using a computer

To produce horizontal lines the − sign or an underline symbol could be used and vertical lines were generally made up by using capital I letters printed one above the other in the same column on the printout

Mosaic graphics symbols

Let us now explore further the set of symbols that can be produced by the Spectrum by running the program shown in Fig. 2.1(a). This produces all of the symbols with character codes from 32 to 164. The codes from 0 to 32 are used for control purposes and do not normally produce symbols on the screen. Codes above 164 are used by the Spectrum's BASIC interpreter to translate the stored program instructions into printed words when it lists a program. To save memory space the actual BASIC commands are stored in memory as single data words known as *tokens*. The character codes above 164 will therefore print out as command words such as PRINT, GOSUB and so on.

```
100 REM Print out character set
110 REM with codes from 32 to 164
120 CLS
130 PRINT
140 FOR n=32 TO 164
150 PRINT CHR$ n;" ";
160 NEXT n
```

Fig. 2.1(a) Program to print the standard character set of the Spectrum.

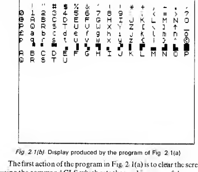


Fig. 2.1(b) Display produced by the program of Fig. 2.1(a).

The first action of the program in Fig. 2.1(a) is to clear the screen by using the command CLS which sets the working area of the screen to white. Next the symbols are printed out by using a simple loop operation and a PRINT CHR$ n command. The CHR$ n command causes the symbol with character code n to be printed on the screen. Note the semicolon at the end of the command line. If this is not included the Spectrum will still print out all of the symbols but it will display one per line and after the first 22 symbols the display will need to be scrolled by pressing the Y key to get the next 22 symbols. The semicolon allows characters from successive PRINT commands to be printed one after another on the same line.

After the set of text symbols you will notice that there is a set of symbols which are made up from small blocks. These are the mosaic graphics symbols. You will see that in these block graphics symbols the symbol space is divided into four segments and each segment may be either black or white giving a total of sixteen different patterns. Each pattern has a character code number and these run from 128 to 143. By simply selecting the appropriate character code using a CHR$ expression in a PRINT statement we can produce any one of the sixteen block patterns as required.

Figure 2.2 shows the complete set of block patterns and their corresponding character codes.

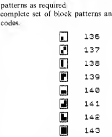


Fig. 2.2 The mosaic block patterns and their associated character codes.

An alternative way of printing the graphics symbols is to use the keyboard in its graphics mode. This mode can be entered from the normal mode by pressing the CAPS SHIFT and 9 keys together. When you are in the graphics mode the flashing cursor on the screen will change from a flashing L to a flashing G. To get back to the normal mode press the CAPS SHIFT and 9 keys again.

When you are in the graphics mode, if the keys in the top row which carry the pictures of the graphics symbols in white, are pressed, then the graphics symbols are printed up on the screen as if they were normal text symbols. There are only eight symbol keys for graphics so to get the second eight graphics patterns the CAPS SHIFT key is pressed as well as the symbol key in the same way as for producing capital letters in text. To print out graphics symbols in a program they are simply enclosed between quotes signs after the PRINT command exactly as if they were normal text symbols.

Adding colour to the display

The initial black and white display of the Spectrum is quite useful for displaying text because it is similar to our familiar black printed text on white paper. However, the Spectrum is capable of much more colourful displays and these are particularly attractive when used with graphics.

To change the colour of the dots making up a symbol on the screen we can use the command INK which is obtained by pressing the CAPS SHIFT and SYMBOL SHIFT keys together to get the *extended* keyboard mode and then pressing the X and CAPS SHIFT keys together to get the INK command. The command INK is followed by a number from 0 to 9 which sets the new INK colour. Any new symbols printed on the screen will now be displayed in the newly selected colour but those symbols already being displayed remain unchanged.

The colours produced by the number following the INK command are as follows:

0	Black	5	Cyan
1	Blue	6	Yellow
2	Red	7	White
3	Magenta	8	Transparent
4	Green	9	Contrast

The colour cyan is a pale blue green colour which is produced effectively by mixing blue and green on the TV screen. Magenta (red + blue) and yellow (red + green) are also produced by mixing red with blue or green.

Colour numbers 8 and 9 are rather different from the others since they do not directly specify a new INK colour. You will remember from Chapter One that the Spectrum allocates colours to each of the symbol spaces and stores this colour information in a separate part of the memory from the display dot patterns. At switch on all character positions will be allocated INK 0 or black as the ink colour. When INK 8 is used the symbol will be printed in whatever ink colour is already allocated to that particular character space on the screen. This can be useful where perhaps different areas of the screen are displayed in different colours. Once the initial display has been set up any new information could be printed using INK 8 and the colour produced will depend upon where the new text is printed on the screen. Thus items can readily be updated without having to decide which ink colour they should have and then switching to a new ink colour.

If a light colour such as yellow is displayed on a white background the text tends to be difficult to read because it is similar to the background colour. If colour 9 is used the computer will automatically choose either white or black text according to whether the background colour is dark or light. Thus the text symbols will always be in contrast to the background so that they are easily read.

To see the effect of colour on the display we can now try running a program which produces simple wallpaper style patterns in a range of colours using the mosaic graphics and some of the text symbols. This program is listed in Fig. 2.3 and a typical pattern is as shown in Fig. 2.4.

```
100 REM Wallpaper patterns
110 DIM a(7)
120 DIM c(7)
130 FOR s=1 TO 30
140 CLS
150 REM Set up pattern
160 FOR p=1 TO 7
170 LET a(p)=122+INT (RND*22)
180 LET c(p)=INT (7*RND)
190 NEXT p
200 REM Print pattern
210 LET k=3+INT (5*RND)
220 FOR n=1 TO 672 STEP k
230 FOR j=1 TO k
240 PRINT  INK c(j);CHR$ a(j);
250 NEXT j
260 NEXT n
270 PAUSE 200
280 NEXT s
```

Fig. 2.3. Program to produce a wallpaper pattern.

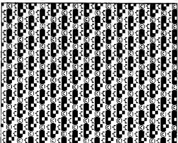


Fig 2.4 Typical display from the wallpaper program

The basic scheme in this program is that the computer generates a random set of seven characters and then it takes three to seven of these as a group and repeatedly prints them in different colours until the screen is filled with a pattern. Each pattern is held on the screen for a few seconds by using a PAUSE command and then the screen is cleared and a new pattern is drawn. The program generates a set of 20 patterns but would go on producing patterns almost indefinitely if the size of the loop using the variables were increased.

Changing the background colour

The Spectrum starts up with a white background, black text and a white border around the display area. If we use the screen clear command CLS this erases any existing displayed symbols and restores the plain white background.

As a change from white we can in fact produce a range of other background colours on the screen by using the PAPER command followed by a CLS command. The PAPER command is produced in a similar way to the INK command except that the C key is used in the extended keyboard mode. The colour numbers after PAPER have exactly the same meanings as the INK colours.

Setting the background using PAPER and CLS clears the screen, so it is important to carry out this operation before starting to print text or draw a picture.

Instead of changing the background colour of the whole screen we can use PAPER to set the background of individual symbols. Thus if we use the command PAPER 2 then all symbols printed after this will have a background colour of red. This red background, however, will appear only in spaces where a new symbol is printed and the background colours of other symbols already being displayed are unaffected.

Colour commands in PRINT statements

Sometimes we may just want to set one or two symbols to a new colour and then return to the normal INK colour. This can of course be done by inserting INK commands to change colour before and after the symbols are printed. A better and more convenient method is to include the INK command in the PRINT statement as in the following:

```
200 PRINT INK2;"Red text"
```

which will temporarily alter the INK colour to red while the message, Red text, is printed, but after this has been done the INK colour will return to whatever INK colour has previously been set up by the program. The same technique can be used with PAPER to change the background colour of a few symbols. Note that when INK or PAPER are used in this way a semicolon must be used to separate the command from the rest of the PRINT statement.

Figure 2.5 shows a modified version of the wallpaper program which uses both INK and PAPER colour changes to give an even more colourful result.

```
100 REM Super wallpaper patterns
110 REM with paper and ink changes
120 DIM a(7)
130 DIM c(7)
140 DIM p(7)
150 FOR s=1 TO 30
160 CLS
170 FOR i=1 TO 7
180 LET a(i)=122+INT (RND*22)
190 LET c(i)=INT (8*RND)
200 LET p(i)=INT (RND*8)
```

```
210 IF p(i)=c(i) THEN  GO TO 185
220 NEXT i
230 REM Print pattern
240 LET k=3+INT (5*RND)
250 FOR n=1 TO 672 STEP k
260 FOR j=1 TO k
270 PRINT  INK c(j); PAPER p(j);CHR$ a(j);
280 NEXT j
290 NEXT n
300 PAUSE 200
310 NEXT s
```

Fig 2.5 A better wallpaper display program

Placing symbols using PRINT AT

So far we have simply printed out rows of symbols on the screen at the point where the text cursor is located. For serious graphics drawing, however, it would be useful if we could place a symbol directly in any of the available spaces on the screen.

In the text mode we can place a text symbol into any one of the available symbol spaces by using the PRINT AT statement which has the form:

100 PRINT AT r,c,"text"

where r and c are the co-ordinates that specify the point on the screen where we want the symbol to be displayed. The value of r can range from 0 to 21 and specifies in which row of the display the text is printed, starting with row 0 at the top of the screen and working down to row 21 at the bottom of the display area. Note that PRINT AT will not allow text to be printed into the bottom two rows (22 and 23) of the screen because these are reserved for use by the computer itself. The variable c indicates how far across the screen the text symbol will start, with 0 at the left edge, going across to 31 at the right edge of the screen.

With 22 rows and 32 columns in each row there are a total of 704 symbol spaces on the screen. Starting at the top left corner the r,c co-ordinates are 0,0. As we move across each row c increases from 0 to 31 and as we move down the screen r increases from 0 to 21 as shown in Fig. 2.6.

The program listed in Fig. 2.7 selects character codes at random and then prints the corresponding symbol at a random position on

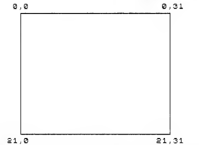


Fig. 2.6 The screen layout for printing using PRINT AT commands

the screen. Here the c and r numbers are generated by multiplying the function RND by 32 and 22 respectively. The RND function produces a number between 0 and 1 although it will never actually produce the number 1. The values for c and r should be integers so we apply the INT operation to the calculated results. The effect of applying INT is to chop off any fractional part of the number leaving us with an integer. Since RND never quite reaches 1 the result for the c calculation will be a number ranging from 0 to 31 which is just what we need for the column position number c. The value of r is similarly rounded down to give a number from 0 to 21.

```
100 REM Printing symbols at random
110 REM positions using PRINT AT
120 FOR n=1 TO 100
125 REM Set row and column position
130 LET r=INT (RND*22)
135 REM Set ink colour
140 LET c=INT (RND*32)
150 INK INT (RND*7)
155 REM Select symbol code
160 LET s=96+INT (RND*48)
165 REM Print symbol on screen
170 PRINT AT r,c;CHR$ s;
180 NEXT n
```

Fig. 2.7 Program to draw symbols at random positions using PRINT AT

Setting individual mosaic points

Since the mosaic graphics displays divides each character space into two rows and two columns we now have a graphics screen which is 64 dots wide and 44 dots high. To use this effectively, however, we must be able to set or reset the states of individual blocks within each character space.

If we examine the coding of the mosaic symbols it will be seen that each of the four blocks in the character space has a numerical value and these are shown in Fig. 2.8. To find the character code for a particular pattern of blocks we simply add together the values of the blocks that are lit (INK colour) and add the result to 128.

Fig. 2.8 The numerical values of the block elements in mosaic graphics symbols.

On our 64 × 44 dot screen we can define the position of a dot by using x to measure its position across the screen starting from the left edge and y for its position down the screen starting from the top. Thus x runs from 0 to 63 and y from 0 to 43.

To find the required row and column position for the mosiac symbol we simply divide y and x respectively by 2 and take the integer values. As an example if we chose x=33 and y=25 the row value r would be:

r = INT(y/2) = INT(12.5) = 12

and the column c would be:

c = INT (x/2) = INT (16.5) = 16

It now remains to choose the correct symbol so that the dot is in the correct place. Starting with the x position if c = x/2 the block needs to be at the left side of the character space and this gives a symbol pattern value of 2. If $c <> x/2$ the block is at the right side and the pattern value is 1. In the y direction if r = y/2 the block is in the upper part of the character space and the pattern value is the one just calculated from the x position. Otherwise the block is in the lower part of the space and the pattern value needs to be multiplied by 4 to give the correct pattern.

To convert from x,y values to a character position and code we can now use the following short routine:

```
500 LET c = INT (x/2)
510 LET r = INT (y/2)
520 LET p = 1
530 IF c = x/2 THEN LET p = p*2
540 IF r <> y/2 THEN LET p = p*4
550 PRINT AT r,c; CHR$ (128+p);
560 RETURN
```

This works fine when we start with a blank screen but problems occur when a new dot is to be printed into a character position where a dot is already being displayed. The new mosaic symbol that we print will effectively erase any symbol already at that position. So we need to be able to check which dots are already lit within the symbol space and then add the new dot if required to produce a new symbol for printing.

Because the Spectrum stores dot patterns, rather than character codes, in its display memory we cannot simply use PEEK to discover which symbol is being displayed as might be done with other types of computer. In fact the Spectrum does have a command called SCREEN$ which will give the character code for the symbol displayed at any character position on the screen. The only problem is that it does not work with the mosaic graphics codes.

All is not lost, however, because we can adopt the same type of scheme used by other computers and set up an array p(c,r) which will store all of the symbol codes that we print to the screen. Now by calling up the appropriate row and column in the array we can find out the code at any character position.

In the array we actually store the value of the blocks in the graphics symbol, which is the character code minus 128. The pattern code for the symbol at our selected character position is checked to see if the dot we want to plot is already lit or not. If it is lit then 128 is added to the pattern code p(c,r) to get the character code for printing. If the dot is not lit the code for that dot is added to p(c,r) and the new value stored and then this code is used to produce the character code for printing.

At this point we can produce a subroutine which will plot a point at any x,y position on the mosaic graphics screen. This is used in the program of Fig. 2.9 to plot a pattern of random dots on the screen.

```
100 REM Random mosaic dots
110 DIM p(32,22)
120 PRINT AT 0,0;"Setting up array";
130 FOR y=1 TO 22: FOR x=1 TO 32
140 LET p(x,y)=0
150 NEXT x: NEXT y: CLS
160 REM Plot dots
170 FOR n=1 TO 500
180 LET x=INT (RND*64)
190 LET y=INT (RND*44)
200 INK INT (RND*8)
210 GO SUB 500
220 NEXT n
230 STOP
490 REM Dot plotting subroutine
500 LET r=INT (y/2)
510 LET c=INT (x/2)
520 LET p1=1
530 IF c=x/2 THEN  LET p1=p1*2
540 IF r<>y/2 THEN  LET p1=p1*4
550 LET p2=p(c+1,r+1)
560 IF p2<8 AND p1=8 THEN  GO TO 640
570 IF p2>=8 THEN  LET p2=p2-8
580 IF p2<4 AND p1=4 THEN  GO TO 640
590 IF p2>=4 THEN  LET p2=p2-4
600 IF p2<2 AND p1=2 THEN  GO TO 640
610 IF p2>=2 THEN  LET p2=p2-2
620 IF p2<1 AND p1=1 THEN  GO TO 640
630 GO TO 650
640 LET p(c+1,r+1)=p(c+1,r+1)+p1
650 PRINT AT r,c;CHR$ (128+p(c+1,r+1));
660 RETURN
```

Fig. 2.9 Program to display random dots using mosaic graphics symbols.

Drawing lines

To draw a line on the low resolution screen we could work out the positions of all the points that need to be lit by first drawing the line on a piece of squared paper representing the screen grid. Each square that the line passes through is one that needs to be lit and its x,y co-ordinates can be read off from the grid. These co-ordinates are then used in the point plotting subroutine to ink in the required points and draw a line on the screen. This is a rather tedious process and it is much easier to let the computer to do the job automatically. All we shall need to tell the computer are the x and y co-ordinates of the two ends of the required line. Drawing horizontal and vertical lines is relatively straightforward.

Suppose we want to draw a horizontal line between two points, x1 and x2. Now because the line is horizontal, the y co-ordinate will be the same at each end of the line. Since each point represents one step in the value of x the number of points we have to plot is determined by the difference between x1 and x2.

Figure 2.10 shows a program which will draw a horizontal line. The change in the value of x between successive dots along the line must be 1 so we can set the x step value xs = 1. If we assume that x1 and x2 can be anywhere on the screen the result of calculating x2 − x1 could be negative and in that case we make xs = −1. The number of points along the line is n = ABS(x2 − x1). Since we are using n in a loop the ABS function is used so that if x2 − x1 is negative n will still be positive.

The line is actually drawn by a simple loop operation where s steps from 1 to ns and on each pass through the loop a point is set on the screen. The x value for each point is calculated by adding s*xs to the value of x1. S starts at 0 so that the first point plotted is at x1.

Drawing a vertical line on the screen follows a similar procedure, but this time the y values of the points change as we move along the line while x stays constant. The program for this is shown in Fig. 2.11. In effect the x and y terms have simply been transposed in the program.

One point to note is that if two lines, drawn with different ink colours, cross one another some points on the original line may change colour. This is because of the restriction that there can be only one ink colour in any character position on the text screen. Remember that although we are setting individual points they are in fact only blocks within the mosaic symbols and a symbol will have all blocks the same colour. If a new point is set within a space where some of the other blocks are already set to INK colour then they will all change colour to the current INK colour.

```
100 REM Horizontal mosaic lines
110 DIM p(32,22)
120 PRINT AT 0,0;"Setting up array";
130 FOR y=1 TO 22: FOR x=1 TO 32
140 LET p(x,y)=0
150 NEXT x: NEXT y: CLS
160 REM Draw lines
170 FOR n=1 TO 50
175 REM Set start and end points
180 LET x1=INT (RND*64)
190 LET x2=INT (RND*64)
200 LET y1=INT (RND*44)
205 REM Set drawing direction
210 LET xs=SGN (x2-x1)
215 REM Set number of steps
220 LET ns=ABS (x2-x1)
230 FOR s=1 TO ns
240 LET x=x1+s*xs
250 LET y=y1
260 GO SUB 500
270 NEXT s
280 INK INT (RND*7)
290 NEXT n
300 STOP
490 REM Dot plotting subroutine
500 LET r=INT (y/2)
510 LET c=INT (x/2)
520 LET p1=1
530 IF c=x/2 THEN  LET p1=p1*2
540 IF r<>y/2 THEN  LET p1=p1*4
550 LET p2=p(c+1,r+1)
560 IF p2<8 AND p1=8 THEN  GO TO 640
570 IF p2>=8 THEN  LET p2=p2-8
580 IF p2<4 AND p1=4 THEN  GO TO 640
590 IF p2>=4 THEN  LET p2=p2-4
600 IF p2<2 AND p1=2 THEN  GO TO 640
610 IF p2>=2 THEN  LET p2=p2-2
620 IF p2<1 AND p1=1 THEN  GO TO 640
630 GO TO 650
640 LET p(c+1,r+1)=p(c+1,r+1)+p1
650 PRINT AT r,c;CHR$ (128+p(c+1,r+1));
660 RETURN
```

Fig 2.10 Program to draw horizontal lines with mosaic symbols

```
100 REM Vertical mosaic lines
110 DIM p(32,22)
120 PRINT AT 0,0;"Setting up array";
130 FOR y=1 TO 22: FOR x=1 TO 32
140 LET p(x,y)=0
150 NEXT x: NEXT y: CLS
160 REM Draw lines
170 FOR n=1 TO 50
175 REM Set start and end points
180 LET x1=INT (RND*64)
190 LET y1=INT (RND*44)
200 LET y2=INT (RND*44)
205 REM Set drawing direction
210 LET ys=SGN (y2-y1)
215 REM Set number of steps
220 LET ns=ABS (y2-y1)
230 FOR s=1 TO ns
240 LET x=x1
250 LET y=y1+s*ys
260 GO SUB 500
270 NEXT s
280 INK INT (RND*7)
290 NEXT n
300 STOP
490 REM Dot plotting subroutine
500 LET r=INT (y/2)
510 LET c=INT (x/2)
520 LET p1=1
530 IF c=x/2 THEN  LET p1=p1*2
540 IF r<>y/2 THEN  LET p1=p1*4
550 LET p2=p(c+1,r+1)
560 IF p2<8 AND p1=8 THEN  GO TO 640
570 IF p2>=8 THEN  LET p2=p2-8
580 IF p2<4 AND p1=4 THEN  GO TO 640
590 IF p2>=4 THEN  LET p2=p2-4
600 IF p2<2 AND p1=2 THEN  GO TO 640
610 IF p2>=2 THEN  LET p2=p2-2
620 IF p2<1 AND p1=1 THEN  GO TO 640
630 GO TO 650
640 LET p(c+1,r+1)=p(c+1,r+1)+p1
650 PRINT AT r,c;CHR$ (128+p(c+1,r+1));
660 RETURN
```

Fig 2.11. Program to draw vertical lines using mosaic graphics

A simple sketching program

Producing pictures on the low resolution screen can be quite a laborious business since it generally involves making the drawing on a suitable piece of squared paper with 32 spaces across and 22 down the sheet. Each square is then subdivided into quarters and the individual blocks within the squares are shaded in to produce the required picture. The symbols across each row are then converted into a string of text symbols and finally printed to the screen to produce the picture.

An alternative technique for producing drawings on the screen is to use a simple sketching program working on similar lines to an 'Etch a Sketch' machine. In such a machine a pen can be moved over the sheet of paper in either a horizontal or vertical direction by means of two knobs or levers. The pen itself can be either held down on the paper to draw a line or lifted up clear as it moves across the sheet.

It is fairly easy to produce a program in which the pen movement around the screen is controlled by using the arrow keys at the top of the Spectrum keyboard. The change of state of the pen between up and down may be controlled by using the U and D keys.

One of the first requirements here is to be able to detect which key on the keyboard has been pressed so that the appropriate action can be taken. This can be done by using the command INKEY$. The INKEY$ command will return a number corresponding to the character code of the key that is being pressed. This command, however, doesn't wait until you press a key: it simply checks if a key is being pressed at the moment it is executed. To monitor the keyboard we need to set up a continuous loop action using the following statement:

```
210 LET a$=INKEY$:IF a$="" THEN GOTO 210
```

Here the string a$ is set to the code produced by INKEY$. If no key is being pressed then a$ will be a blank string ("") and the instruction loops back to itself and repeats continuously. When a key is pressed a$ is not blank and the test fails so that the program now moves on to the next instruction.

Having detected the key press we now have to examine the value of a$ to find out which key was pressed. We can start by checking for the arrow keys. The actual codes for the cursor shift operation of these keys is obtained only when they are operated in conjunction with the CAPS SHIFT key. On the Spectrum the four arrow keys are also the number keys 5, 6, 7 and 8. In the normal keyboard mode these keys will therefore produce the codes for the numbers 5, 6, 7 or 8 as follows:

Left arrow	5
Right arrow	8
Down arrow	6
Up arrow	7

If right arrow is detected the value for x is increased by 1 while for a left arrow x is reduced by 1. The y value is increased if the down arrow is detected and decreased for up arrow. Tests are made to detect x values less than 0 or greater than 63 since these would cause errors in the PRINT AT command. If x is less than 0 it is set to 0 and if greater than 63 it is set at 63. This gives a cutoff effect so that the line stops at the side of the screen. Similar checks are made on the y values to give vertical limiting.

If the D key is detected a variable w is set at 1 to show that the pen is down and if U is pressed w is set to 0 to indicate that the pen is up.

```
100 REM Sketching program
110 REM for mosaic graphics
120 PRINT AT 0,0:"Setting up screen array";
130 DIM p(32.22)
140 FOR y=1 TO 22: FOR x=1 TO 32
150 LET p(x,y)=0
160 NEXT x: NEXT y: CLS
170 LET x1=32: LET x2=32
180 LET y1=22: LET y2=22
190 LET w=1: LET x=x1: LET y=y1: GO SUB 700
195 REM Sketching loop
200 PRINT AT 0,0;"x=";x1;" y=";y1;"      ";
210 LET a$=INKEY$: IF a$="" THEN  GO TO 210
220 IF a$="5" THEN  LET x1=x1-1: GO TO 300
230 IF a$="8" THEN  LET x1=x1+1: GO TO 300
240 IF a$="6" THEN  LET y1=y1+1: GO TO 300
250 IF a$="7" THEN  LET y1=y1-1: GO TO 300
260 IF a$="d" THEN  LET w=1: GO TO 300
270 IF a$="u" THEN  LET w=0: GO TO 300
280 IF a$="q" THEN  STOP
290 GO TO 200
295 REM Check x,y limits
300 IF x1<0 THEN  LET x1=0
310 IF x1>63 THEN  LET x1=63
320 IF y1<2 THEN  LET y1=2
```

```
 330 IF y1>43 THEN  LET y1=43
 340 LET lp=pc
 350 LET x=x1: LET y=y1: GO SUB 700
 360 LET x=x2: LET y=y2
 370 IF w=0 AND lp=1 THEN  GO SUB 900
 380 LET x2=x1: LET y2=y1
 390 GO TO 200
 690>REM Set dot subroutine
 700 LET r=INT (INT (y)/2)
 710 LET c=INT (INT (x)/2)
 720 LET p1=1: LET pc=0
 730 IF c=INT (x)/2 THEN  LET p1=p1*2
 740 IF r<>INT (y)/2 THEN  LET p1=p1*4
 750 LET p2=p(c+1,r+1)
 760 IF p2<8 AND p1=8 THEN  GO TO 840
 770 IF p2>=8 THEN  LET p2=p2-8
 780 IF p2<4 AND p1=4 THEN  GO TO 840
 790 IF p2>=4 THEN  LET p2=p2-4
 800 IF p2<2 AND p1=2 THEN  GO TO 840
 810 IF p2>=2 THEN  LET p2=p2-2
 820 IF p2<1 AND p1=1 THEN  GO TO 840
 830 GO TO 850
 840 LET p(c+1,r+1)=p(c+1,r+1)+p1
 845 LET pc=1
 850 PRINT AT r,c;CHR$ (128+p(c+1,r+1));
 860 RETURN
 895 REM Erase dot subroutine
 900 LET r=INT (INT (y)/2)
 910 LET c=INT (INT (x)/2)
 920 LET p1=1
 930 IF c=INT (x)/2 THEN  LET p1=p1*2
 940 IF r<>INT (y)/2 THEN  LET p1=p1*4
 950 LET p2=p(c+1,r+1)
 960 IF p2>=8 AND p1=8 THEN  GO TO 1040
 970 IF p2>=8 THEN  LET p2=p2-8
 980 IF p2>=4 AND p1=4 THEN  GO TO 1040
 990 IF p2>=4 THEN  LET p2=p2-4
1000 IF p2>=2 AND p1=2 THEN  GO TO 1040
1010 IF p2>=2 THEN  LET p2=p2-2
1020 IF p2>=1 AND p1=1 THEN  GO TO 1040
1030 GO TO 1050
1040 LET p(c+1,r+1)=p(c+1,r+1)-p1
1050 PRINT AT r,c;CHR$ (128+p(c+1,r+1));
1060 RETURN
```

Fig. 2.12 A simple sketching program for mosaic graphics

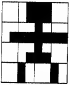
If the pen is down, a point is plotted at the new position by the subroutine at line 5ØØ. If the pen is up the subroutine is skipped or the dot is erased. The x and y position for the pen is printed at the top of the screen. The program listing is shown in Fig. 2.12.

Producing pictures

So far we have made patterns, set individual points, drawn lines and sketched on the screen but often you will want to produce simple pictures. This is best done by laying out the picture on a ruled grid and then working out which symbols have to be printed to produce the required result.

Let us suppose we want to produce a drawing of a little matchstick man. We shall start by deciding to draw the picture on an eight by eight dot (four by four mosaic symbols) area of the screen and so a simple grid is drawn up as shown in Fig. 2.13. The shape of the man is then built up by simply filling in a pattern of blocks within the eight by eight array.

```
Picture            Character codes
```



```
128+133+143+128
132+140+142+140
128+132+142+128
128+138+128+138
```

Fig. 2.13. The mosaic symbols needed to produce a man shaped figure.

Having produced the pattern on the grid we have to turn it into a series of character codes for printing to the screen. Remember that each character consists of four block elements arranged as a 2 × 2 array so we can now mark out on our grid the actual character spaces. Next we can start with the top two rows of the pattern and convert the block patterns into four symbol codes which are shown alongside the grid. The same process is repeated for the other rows in the picture so that we have a total of sixteen character codes in four groups of four.

The character codes are now stored in a 4 × 4 array variable called a This is set up by using a READ loop and a set of DATA statements. Once the array is set up all we have to do is decide where we want the man drawn on the screen. If we want the man to be at row 10 and column 15 then we simply set variables r and c to 10 and 15 respectively. Note that these co-ordinates indicate where the top left corner of the pattern will be printed.

Having chosen the position the man can be printed by simply using two loops which run from 1 to 4 and these call up the elements of the array a as required. The variables for these loops are i, which counts off column positions, and j, which counts off row positions. To get the row and column positions for the PRINT AT statement we simply add j to r and i to c. Since the i and j values start at 1, to get the correct position on the screen, 1 should be subtracted from the r+j and c+i values. Note the semicolon at the end of the PRINT AT line which allows the next symbol to be printed in the following space. The final program to print out little matchstick man becomes as shown in Fig. 2.14.

```
100 REM Figure of a man
110 REM using mosaic graphics
120 DIM a(4,4)
125 REM Set up mosaic symbols
130 FOR j=1 TO 4
140 FOR i=1 TO 4
150 READ a(i,j)
160 NEXT i
170 NEXT j
180 DATA 128,133,143,128
190 DATA 132,140,142,140
200 DATA 128,132,142,128
210 DATA 128,138,128,138
215 REM Print man on screen
220 LET r=10
230 LET c=15
240 FOR j=1 TO 4
250 FOR i=1 TO 4
260 PRINT AT r+j-1,c+i-1;CHR$ a(i,j);
270 NEXT i
280 NEXT j
290 STOP
```

Fig. 2.14 Program to draw man using mosaic graphics.

If we can draw one matchstick man we can equally well draw lots of them all over the screen. In the program shown in Fig. 2.15 this has been done, starting with the first man at position 0,0 at the top

```
100 REM Multiple man figures
110 REM using mosaic graphics
120 DIM a(4,4)
125 REM Set up mosaic symbols
130 FOR j=1 TO 4
140 FOR i=1 TO 4
150 READ a(i,j)
160 NEXT i
170 NEXT j
180 DATA 128,133,143,128
190 DATA 132,140,142,140
200 DATA 128,132,142,128
210 DATA 128,138,128,138
215 REM Print men on screen
220 FOR r=0 TO 16 STEP 5
230 FOR c=0 TO 28 STEP 4
240 FOR j=1 TO 4
250 FOR i=1 TO 4
260 PRINT AT r+j-1,c+i-1;CHR$ a(i,j);
270 NEXT i
280 NEXT j
290 NEXT c
300 NEXT r
310 STOP
```

Fig 2.15 Program to draw multiple men figures all over the screen.

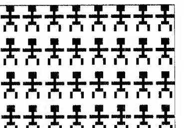


Fig 2.16 Display produced by the program of Fig. 2.15

left of the screen. A point to note here is that after each man has been drawn c is increased by 4 and after each row of men has been drawn the value of r is increased by 5 to leave one blank row on the screen between each row of men

Of course the matchstick men can be drawn at random positions on the screen by using PRINT AT, but if this is done the maximum value of c must be limited to 28 and the maximum value of r to 16 so as to prevent the values of c and r for the lower right character of the set going beyond the screen limits which would cause an error. Here c and r are the column and row positions for the top left character in the group used to draw the figure.

Chapter Three
High Resolution Graphics

For serious picture drawing the low resolution graphics modes of the Spectrum are just not good enough, so we need to move to a much higher level of graphics resolution. As explained in Chapter One, the high resolution mode allows the state of each individual dot in each character space to be controlled. Each character space on the Spectrum screen consists of an 8 × 8 array of dots, so with 32 characters in each row there are 256 dots across the screen. In the vertical direction the bottom two text rows are reserved for the display of commands, so the high resolution screen is limited to 22 text rows giving 176 dots in the vertical direction. Thus the screen layout in high resolution is 256 × 176.

In the computer memory the pattern of states for eight adjacent dots in a row is stored as one eight bit word, so the total memory used for high resolution is 256 × 176 divided by eight, or 5632 bytes of memory. In this scheme each dot can be either 'on' or 'off' or, in other words, it has either the INK (on) colour or the PAPER (off) colour. Colours are set up by using the INK and PAPER commands in the same way as for normal text or mosaic graphics displays.

The colour information is stored in a different area of memory and one pair of colours is allocated to each character space on the screen. This can present some problems as we shall see later, but it does allow all of the colours to be used on the screen at the same time. In many other computers the colour information is stored for each individual pixel. Two data bits are needed for each pixel if four colours are to be used and three bits are needed for eight colours, so the memory becomes twice, or three times, the size required for a black and white display. To reduce the amount of memory needed for the display the number of colours that can be used at a time is often limited to two or four. Computers using this technique do permit adjacent dots to be set to different colours, which may not always be possible on the Spectrum as we shall see.

Setting and resetting individual points

In the high resolution mode we can easily set an individual point on the screen to a specified colour. The command for this is PLOT and the command word is obtained by using the Q key when the flashing K cursor is being displayed. The complete command takes the form:

```
100 PLOT x,y
```

where x and y are the co-ordinates of the point to be plotted. The selected point on the screen will then be set to the current INK colour.

On the high resolution screen there are many more points than we had on the low resolution screen, so the values of x and y can be much larger. In fact, for the high resolution mode the x and y co-ordinates are based upon a 256 × 176 grid. Thus x can range in value from 0 to 255 and y can be from 0 to 175. The value of x is 0 at the left-hand side of the screen and increases to 255 at the right-hand edge. Unlike the text screen, the value for y is 0 at the bottom of the picture and increases to 175 at the top of the screen. Thus the top left corner is point 0,175 and the bottom right corner is point 255,0. The general arrangement is shown in Fig. 3.1.

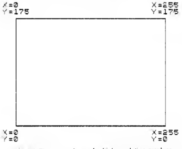


Fig. 3.1 The screen layout for high resolution graphics

Try running the program listed in Fig. 3.2. This program selects random values for x and y and then plots points at random positions all over the screen.

```
100 REM Random dots
110 FOR n=1 TO 1000
120 LET x=INT (RND*256)
130 LET y=INT (RND*176)
140 PLOT x,y
150 NEXT n
```

Fig 3.2 Program to produce random dots in high resolution.

As in the text mode, the Spectrum keeps track of the position on the screen by means of a cursor. The text cursor is generally displayed as a flashing space but the graphics cursor is not displayed, although, the computer does keep the current x,y position of the cursor stored in memory. Whenever a high resolution drawing operation is executed the cursor position held in memory will be updated.

When the Spectrum is switched on, or after the screen has been cleared by using CLS, the cursor is automatically placed at position x,y = 0,0. When a PLOT command is executed the cursor position is set directly to the value of x,y specified in the PLOT command. So by using a PLOT command we can position the cursor wherever we want it on the screen. In virtually all high resolution drawings therefore the first command will be a PLOT to position the cursor.

To reset a point on the screen the command INVERSE 1 is used before the PLOT command. This INVERSE command causes the INK and PAPER colours to be swapped so that the dot is now written in the PAPER colour that is already set at that point. Since the dot is now in PAPER colour it disappears. Other dots in the same character space are unaffected. After the PLOT, or perhaps a series of PLOT commands, the command INVERSE Ø must be used to restore normal operation. The INVERSE operation may be included in the PLOT command as follows:

15Ø PLOT INVERSE1; x,y:INVERSEØ

We shall look more closely at INVERSE in Chapter Six.

Selecting graphics colour

At start up the Spectrum will be set up to produce black text on a white background. High resolution dots and lines will also be displayed as black on a white background. The drawing colour for high resolution is controlled by the INK command in exactly the same way as for text displays. The PAPER colour is not altered by the PLOT command so the dot will appear against whatever PAPER colour already exists around that point on the screen. Try running the program listed in Fig. 3.3 which plots dots at random positions on the screen and with a random INK colour for each dot.

```
100 REM Random coloured dots
110 FOR n=1 TO 10000
120 LET i=INT (RND*7)
130 LET x=RND*255
140 LET y=RND*175
150 INK i
160 PLOT x,y
170 NEXT n
```

Fig. 3.3 Program to produce random coloured dots

As the picture builds up you will notice that at first individual dots appear but later whole groups of dots change colour together. Eventually you will see that the colours form a pattern corresponding to the text character spaces on the screen. This is because the Spectrum stores its colour information separately from the dot pattern. The colour information is stored in terms of text character positions and the Spectrum will allow only one pair of colours in any text character space on the screen. Thus when a new dot is plotted the INK colour for the character space in which the dot is located is set to the current INK colour. If there are any other dots already displayed in that character space then they will change colour to the colour of the new dot. This can impose some limitations on the production of colour pictures using the Spectrum since adjacent dots cannot always be plotted in different colours.

Mixing text and graphics

On some home computers the text and high resolution graphics use different display modes and it can sometimes be difficult to mix text and graphics on the same display. Since the Spectrum stores text symbols as dot patterns in its display memory there is no difficulty in mixing text and graphics on the screen. The text symbols are produced by simply using PRINT or PRINT AT and graphics points may be plotted over the text symbols by using PLOT commands.

When mixing text and graphics, however, it is important to note that whereas on the text screen the rows are numbered from top to bottom with row 0 at the top, when we use graphics the y co-ordinates are numbered with 0 at the bottom of the screen and y increases as we move up the screen. Since each symbol is eight dots wide and eight dots high the row and column values are easily converted into x and y values by simply multiplying them by 8. Thus to calculate x we could use the equation

x = 8 * c

To calculate y we need to correct for the difference in layout of the r an y values which is done by using

y=175−8*r

to calculate the value for y.

To convert from x,y to r,c values we simply have to divide x and y by 8 and then round them down by taking the integer value using the INT command. Note here that the INT command simply chops off the fractional part of a number. Again a correction must be made for the different layout of y and r so the calculations are as follows:

r=INT((175−y)/8)
c = INT (x/8)

A point to note here is that unlike the PRINT AT command the PLOT command has the horizontal co-ordinate (x) first.

Drawing lines

Setting points is all very well, but for most purposes we shall want to draw lines on the screen. Whereas in the low resolution mode we had to do this by calculating the points that had to be set and then setting them, it is much easier in the high resolution mode because there is a special line drawing command. This command is DRAW and is obtained by using the W key when the Spectrum is displaying the flashing K cursor. The form of this command is:

```
100 DRAW x,y
```

There is a very important difference between the x,y values used with DRAW and those used for PLOT. In the case of PLOT the x,y values specify the actual position on the screen. For the DRAW command the x and y values are measured relative to the current position of the graphics cursor. As an example, suppose we used the command:

```
100 PLOT 30,20
```

This instruction will set the cursor at position x=30,y=20 on the screen and then place a dot at that position.

If we now use the command:

```
110 DRAW 30,20
```

a line will be drawn from the point we have just plotted to a new point 30 units to the right and 20 units up from that position. The cursor now moves to the end of the line that has just been drawn. This means that the cursor is now at a position where x = 60 (i.e. 30 + 30) and where y = 40 (i.e. 20 + 20).

This method of calculating positions on the screen is known as relative plotting and can often make the drawing of shapes easier since we no longer have to calculate the absolute x and y co-ordinates for each point in the shape.

A point to note here is that both x and y in a DRAW statement can be negative. A negative value for x simply means that the line is drawn to the left from the current position while a negative y value will cause the line to be drawn down from the current position.

As in the case of plotting individual dots using PLOT, the lines produced by using DRAW are displayed in the current INK colour.

Drawing lines between specified points

In many cases we shall want to be able to draw a line between two specific points on the screen. Let us assume that these points are given by the co-ordinates x1,y1 and x2,y2. Since the DRAW command uses relative co-ordinates some calculations are required to obtain the x,y parameters for the DRAW command.

The first step in drawing the line is to place the cursor at one end of the line, say at point x1,y1. This is easily done by using PLOT x1,y1 to place a dot at the chosen point. Next we have to calculate the x,y values for the DRAW command. To do this we must find the difference between x1 and x2 and also between y1 and y2. Thus x=x2−x1 and y=y2−y1. There is no need to actually calculate values of x and y separately since this can easily be done in the actual DRAW statement by replacing the x and y terms with expressions as follows:

```
100 DRAW x2-x1,y2-y1
```

In this statement the value of the x term becomes negative when x2 is to the left of x1, and the y term becomes negative if y2 is below y1 on the screen.

Having drawn a line the cursor will have moved to the far end of the line (x2,y2) in this case. Now if we want to draw a further line starting from the end of the line just drawn there is no need for a PLOT command since the cursor is already in position.

Making ribbon patterns

Now that we can draw lines and control the colour of our display it becomes possible to experiment with producing patterns on the screen. For a start we might generate a simple moving line pattern combined with colour changes.

The principle involved in drawing this type of pattern is that we start by choosing two random points (x1,y1 and x2,y2) on the screen and then draw a line between them. Next we alter the two points by a small amount and then draw another line. The pattern then builds up as more lines are drawn with a small change to the values of x1,y1 and x2,y2 before each new line is drawn.

If the same change in values x and y is made at both ends of the line then its length will remain constant and as successive lines are drawn a ribbon of colour is produced on the screen. If different amounts of change are made at the opposite ends of the line, the width of the ribbon changes and it also curls as it moves around the screen.

When the computer is working out x,y co-ordinates it is possible that the values of x and y produced may fall outside the permissible range. Some computers can tolerate this state of affairs and will carry out a wraparound operation. Thus a point that goes off the right side of the screen is automatically corrected and reinserted near the left side of the screen. The Spectrum, however, will merely stop the program and signal an error if either x or y goes out of range.

To deal with this possible error condition we need to test the calculated values of x and y before using them to draw a line on the screen. This is done by simply checking to see if x is greater than 255 or less than 0 and a similar test is carried out on y. In the pattern drawing program when one of the points reaches an edge of the screen the appropriate increment for x or y is reversed in sign and the co-ordinates recalculated. This has the effect of sending the line back across the screen as if it had been reflected off the edge of the screen.

```
100 REM Moving line pattern
110 FOR n=1 TO 20
120 LET dx1=2-INT (5*RND)
130 LET dx2=2-INT (5*RND)
140 LET dy1=2-INT (5*RND)
150 LET dy2=2-INT (5*RND)
160 LET x1=20+RND*200
170 LET y1=20+RND*130
180 LET x2=20+RND*200
190 LET y2=20+RND*130
200 INK INT (7*RND)
210 PAPER 7
220 CLS
230 FOR k=1 TO 500
240 PLOT x1,y1
250 DRAW x2-x1,y2-y1
260 LET x1=x1+dx1
270 IF x1<=255 AND x1>=0 THEN  GO TO 310
280 LET dx1=-dx1
290 LET x1=x1+dx1
300 INK INT (RND*7)
310 LET x2=x2+dx2
320 IF x2<=255 AND x2>=0 THEN  GO TO 360
330 LET dx2=-dx2
340 LET x2=x2+dx2
350 INK INT (RND*7)
360 LET y1=y1+dy1
370 IF y1<=175 AND y1>=0 THEN  GO TO 410
380 LET dy1=-dy1
390 LET y1=y1+dy1
400 INK INT (RND*7)
410 LET y2=y2+dy2
420 IF y2<=175 AND y2>=0 THEN  GO TO 460
430 LET dy2=-dy2
440 LET y2=y2+dy2
450 INK INT (RND*7)
460 NEXT k
470 NEXT n
```

Fig. 3.4. Program to produce a ribbon type pattern.

A pattern drawing program of this type is listed in Fig. 3.4. To make the picture more colourful a different colour is selected each time the line reaches one of the sides of the screen.

This program also demonstrates the limitations imposed by the Spectrum's method of storing the INK and PAPER colour information. Because the colour controls a complete symbol space, if a different ink colour is used in a space where some dots are already lit then all of the dots in that space will change colour to the new ink colour. This produces a stepped effect on the display as the pattern is drawn over a part of the pattern that was in a different colour. Despite this slight limitation the program will still produce some quite attractive abstract patterns.

Producing moiré patterns

Another quite attractive type of pattern that can be produced by using high resolution graphics is the moiré pattern. It is easy to produce such patterns by simply choosing a random origin point on the screen and then drawing a pattern of radial lines from that point out to the edges of the screen. By varying the pitch of the lines and the position of the origin point we can generate a selection of patterns which resemble those sometimes seen on silk or taffeta materials. The program listed in Fig. 3.5 uses random functions to generate the origins of the patterns and their pitch. Figure 3.6 shows a typical pattern as produced on the screen.

```
100 REM Moire patterns
110 INK 0: PAPER 7
120 FOR k=1 TO 100
130 CLS
140 LET cx=20+200*RND
150 LET cy=20+130*RND
160 LET s=2+INT (3*RND)
170 FOR x=0 TO 255 STEP s
180 PLOT cx,cy
190 DRAW x-cx,0-cy
200 PLOT cx,cy
210 DRAW x-cx,175-cy
220 NEXT x
230 FOR y=0 TO 175 STEP s
240 PLOT cx,cy
250 DRAW 0-cx,y-cy
260 PLOT cx,cy
270 DRAW 255-cx,y-cy
280 NEXT y
290 PAUSE 500
300 NEXT k
```

Fig. 3.5. Program to produce a simple moiré pattern.

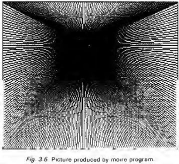


Fig. 3.6 Picture produced by moire program.

We can produce much more colourful patterns by setting up random INK and PAPER colours before drawing the pattern as shown in the program listed in Fig. 3.7. A CLS command after the PAPER command sets the whole screen to the selected PAPER colour and then the pattern is drawn on top in the selected INK colour. In this program a random selection of INK, PAPER and BORDER colours is set up for each new pattern.

```
100 REM Coloured moire patterns
110 FOR k=1 TO 100
120 LET cx=20+200*RND
130 LET cy=20+130*RND
140 LET p=INT (RND*8)
150 LET i=INT (RND*8)
160 IF p=i THEN GO TO 150
170 PAPER p
180 INK i
190 BORDER INT (RND*8)
200 CLS
210 LET s=2+INT (3*RND)
220 FOR x=0 TO 255 STEP s
230 PLOT cx,cy
240 DRAW x-cx,0-cy
250 PLOT cx,cy
260 DRAW x-cx,175-cy
```

```
270 NEXT x
280 FOR y=0 TO 175 STEP s
290 PLOT cx,cy
300 DRAW 0-cx,y-cy
310 PLOT cx,cy
320 DRAW 255-cx,y-cy
330 NEXT y
340 PAUSE 500
350 NEXT k
```

Fig 3.7 Program for moiré patterns in colour

Drawing dotted lines

The line drawing command will draw a solid line between points x1,y1 and x2,y2, but suppose we wanted to draw a dotted line between these two points. There is no convenient command on the Spectrum for this task so we must fall back on a line drawing routine which calculates and plots individual points to make up the line.

The dotted line is drawn by calculating which pixels along the line must be lit and then lighting them by using the PLOT command. The basic line drawing routine calculates the differences between x1 and x2 and between y1 and y2 then takes the larger of these differences as the number of points to be plotted.

The next step is to calculate the increments of x and y between successive points. If the larger number of points is in the x direction then the x increment is set at 1 and the y increment is a fraction calculated by dividing the difference between y2 and y1 by the total number of points np. If the y direction has a larger number of points the y increment is 1 and the x increment becomes less than 1.

For a dotted line we need to plot alternate points along the line so the plotting loop steps 2 units at a time. At the end of the plotting loop a single additional point is plotted at point x2,y2 to ensure that the line is of the correct length. In the program shown in Fig. 3.8 the line drawing routine has been made into a subroutine and the main program draws a series of dotted lines between random points on the screen. The point to note here is that the values of x1,y1 and x2,y2 representing the ends of the line must be set up in the main program before the subroutine is called.

A dashed line or even a line with alternate dots and dashes could also be drawn by modifying the drawing subroutine. For a dashed line the routine would step forward three pixels at a time but in this

```
100 REM Dotted lines
110 FOR n=1 TO 20
120 LET x1=INT (RND*255)
130 LET x2=INT (RND*255)
140 LET y1=INT (RND*175)
150 LET y2=INT (RND*175)
160 LET xs=1
170 LET ys=1
180 LET xi=1
190 LET yi=1
200 LET dx=x2-x1
210 LET dy=y2-y1
220 IF dx<0 THEN  LET xs=-1
230 IF dy<0 THEN  LET ys=-1
240 LET nx=ABS (dx)
250 LET ny=ABS (dy)
260 IF nx>=ny THEN  LET np=nx: LET
    yi=ny/nx
270 IF ny>nx THEN  LET np=ny: LET
    xi=nx/ny
280 PLOT x1,y1
290 LET s=INT (RND*3)+2
300 FOR j=0 TO np STEP s
310 LET x=x1+xs*INT (j*xi+.5)
320 LET y=y1+ys*INT (j*yi+.5)
330 PLOT x,y
340 NEXT j
350 PLOT x2,y2
360 NEXT n
```

Fig 3.8 Program to draw dotted lines

case two points have to be set on each pass through the drawing subroutine. I will leave you to experiment with this for yourself.

Drawing triangles

Perhaps the simplest figure or shape that we can draw is the triangle which has three sides and three corners. To draw the triangle we need to know the screen position co-ordinates of the three corners which we shall call x1,y1, x2,y2 and x3,y3 as shown in Fig. 3.9. The process of drawing the triangle involves drawing a line from x1,y1 to x2,y2 then a second line from x2,y2 to x3,y3. Finally, the triangle is completed by drawing a line from x3,y3 to x1,y1.

When we come to the process of actually drawing the triangle the

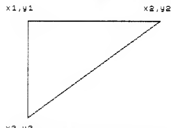


Fig 3 9 The drawing co-ordinates for a triangle

first step is to position the graphics cursor at one of the corners such as point x1,y1 To do this a single point is placed at x1,y1 by using a PLOT statement which positions the cursor ready for drawing the first line The three sides of the triangle are then drawn by using three DRAW commands Figure 3 10 shows a simple program which selects random sets of three points and then draws triangles through them Here the triangle drawing operation is written as a subroutine

```
100 REM Random triangle drawing program
110 FOR s=1 TO 20
120 BORDER INT (RND*8)
130 LET p=INT (RND*8)
140 PAPER p: CLS
150 FOR n=1 TO 15
160 REM Set corner points
170 LET x1=INT (RND*255)
180 LET y1=INT (RND*175)
190 LET x2=INT (RND*255)
200 LET y2=INT (RND*175)
210 LET x3=INT (RND*255)
220 LET y3=INT (RND*175)
230 REM Set ink colour
240 LET c=INT (RND*8)
250 IF c=p THEN  GO TO 190
260 INK c
270 REM Draw triangle
280 PLOT x1,y1
```

```
290 DRAW x2-x1,y2-y1
300 DRAW x3-x2,y3-y2
310 DRAW x1-x3,y1-y3
320 NEXT n
330 PAUSE 200
340 NEXT s
350 INK 0
```

Fig 3.10 A random triangle drawing program.

Mirror image patterns

Producing random triangles gives quite interesting patterns but we can get more attractive results by using the mirror image principle. In this case four mirror image patterns are drawn around the centre point of the screen so that each pattern fills a quarter of the screen.

The technique involved is to draw a random shape triangle in the upper right quarter of the screen by adding the x,y co-ordinates of the first point of the triangle to the x,y co-ordinates at the centre of the screen and the triangle is drawn using three DRAW commands as in the last program. The same basic triangle is then drawn again, but this time the y co-ordinate of its first point is subtracted from the screen centre co-ordinates so that the triangle is positioned below the centre line of the screen. To turn the triangle upside-down, the y co-ordinates in the three DRAW statements are transposed so that y2–y1 becomes y1–y2 and so on. For the other two quarters of the screen the same two sequences are used but here the starting x value is subtracted from the centre point x co-ordinate and the x terms are transposed in the three DRAW commands to place the triangles to the left of the centre and turn them around from left to right.

Figure 3.11 gives the listing for a program to create this type of pattern which resembles the patterns produced in a kaleidoscope. In the program random colours are used in a series of triangles which build up the pattern, and successive patterns are produced with randomly selected background and border colours. The result on the screen is similar to that shown in Fig. 3.12 but very colourful.

```
100 REM Kaleidoscope program
110 FOR s=1 TO 20
120 BORDER INT (RND*8)
130 LET p=INT (RND*8)
140 PAPER p: CLS
150 FOR n=1 TO 20
```

```
160 REM Set corner points
170 LET x1=INT (RND*127)
180 LET y1=INT (RND*87)
190 LET x2=INT (RND*127)
200 LET y2=INT (RND*87)
210 LET x3=INT (RND*127)
220 LET y3=INT (RND*87)
230 REM Set ink colour
240 LET c=INT (RND*8)
250 IF c=p THEN GO TO 190
260 INK c
270 REM Draw triangles
280 PLOT 128+x1,88+y1
290 DRAW x2-x1,y2-y1
300 DRAW x3-x2,y3-y2
310 DRAW x1-x3,y1-y3
320 PLOT 128+x1,88-y1
330 DRAW x2-x1,y1-y2
340 DRAW x3-x2,y2-y3
350 DRAW x1-x3,y3-y1
360 PLOT 128-x1,88-y1
370 DRAW x1-x2,y1-y2
380 DRAW x2-x3,y2-y3
390 DRAW x3-x1,y3-y1
400 PLOT 128-x1,88+y1
410 DRAW x1-x2,y2-y1
420 DRAW x2-x3,y3-y2
430 DRAW x3-x1,y1-y3
440 NEXT n
450 PAUSE 200
460 NEXT s
470 INK 0
```

Fig 3 11 Simple kaleidoscope program

Drawing rectangles and squares

Let us now move on to draw a figure with four sides and corners, which is a rectangle. The simplest approach is to work out the x,y co-ordinates for the four corners of the rectangle and then to draw four lines which link the points together to form the sides of the rectangle. As an example, we might wish to draw a rectangle which is 100 units wide and 50 units high and we can choose a position (x1,y1) of say 40,50 for the position of the bottom left corner. The

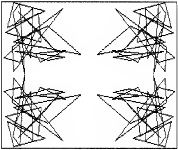


Fig 3 12 Typical kaleidoscope picture

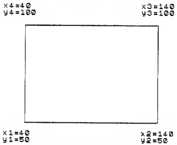


Fig 3 13 Drawing co-ordinates for a rectangle

values for the other corners are shown in Fig. 3.13. Now all we need to do is use a PLOT command to place the cursor at the bottom left corner and then four DRAW commands to actually draw the four lines that make up the rectangle. This is shown in Fig. 3.14.

```
100 REM Rectangle drawing using
110 REM corner x,y coordinates
120 LET x1=40
130 LET y1=50
140 LET x2=140
150 LET y2=50
160 LET x3=140
170 LET y3=100
180 LET x4=40
190 LET y4=100
200 PLOT x1,y1
210 DRAW x2-x1,y2-y1
220 DRAW x3-x2,y3-y2
230 DRAW x4-x3,y4-y3
240 DRAW x1-x4,y1-y4
```

Fig 3.14 Program to draw a rectangle using corner co-ordinates

Using width and height

Drawing a rectangle by using the corner co-ordinates is not the best way of making use of the Spectrum's DRAW command. Rectangles have a width, w, and a height, h, as shown in Fig. 3.15 and by using these we can take advantage of the relative plotting scheme used by the DRAW command. In this case we need only supply the co-ordinates of one corner and the values for w and h of the rectangle. A

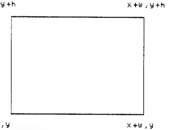


Fig 3.15 Rectangle related to its height and width

PLOT command is used to place a dot and the cursor at the bottom left corner of the rectangle. The first line is along the bottom side of the rectangle and is horizontal, so the y value for the DRAW command is 0. The length of the line is w units so the x value in the DRAW command must be equal to w. The first DRAW statement therefore becomes:

```
110 DRAW w,0
```

The next line is the right-hand side which is vertical and h units long. Here the x term of the DRAW command is 0 since the line is vertical and the y term will be equal to the value h, so the command becomes:

```
120 DRAW 0,h
```

The top side of the rectangle is again w units long, but this time the line is drawn from right to left so the x term must be negative and in fact has a value of −w, whilst the y term is 0. The final line is vertical, so the x term is 0 and the y term is −h since the line is drawn down the screen. This is shown in the program listed in Fig. 3.16.

```
100 REM Rectangle drawing
110 REM using width and height
120 LET x=50
130 LET y=50
140 LET w=100
150 LET h=50
160 REM Plot lower right corner
170 PLOT x,y
180 REM Draw bottom side
190 DRAW w,0
200 REM Draw right side
210 DRAW 0,h
220 REM Draw top side
230 DRAW -w,0
240 REM Draw left side
250 DRAW 0,-h
```

Fig. 3.16 Rectangle drawing program using height and width

A square is just a special version of a rectangle where the height h and the width w are equal. To draw a square we could use either the basic rectangle drawing routine by setting h=w or a different routine using only one variable w. In this case the h terms in the rectangle drawing routine are simply replaced by w terms.

Dealing with screen limits

If we try to draw a rectangle 100 units wide with its lower corner at an x position of say 200, the program will fail and a message indicating an out of range integer will appear on the screen. This is because the value of the x co-ordinate has gone beyond its maximum permissible value which is 255. In some home computers this condition is automatically dealt with by the line drawing command so that the line is either drawn just to the screen limit or is wrapped around so that the part of the line that would have been off the screen is drawn at the other side. An important point to remember is that the line drawing command of the Spectrum has no built-in correction for this situation.

```
100 REM Random rectangles
110 REM with screen edge clipping
120 FOR s=1 TO 20
130 BORDER INT (8*RND)
140 LET p=INT (8*RND)
150 PAPER p
160 CLS
170 FOR n=1 TO 10
180 REM Set rectangle position and size
190 LET x=INT (240*RND)
200 LET y=INT (165*RND)
210 LET w=10+INT (100*RND)
220 LET h=10+INT (100*RND)
230 LET c=INT (8*RND)
240 IF c=p THEN  GO TO 230
250 INK c
260 REM Clip rectangle at screen edge
270 IF x+w>255 THEN  LET w=255-x
280 IF y+h>175 THEN  LET h=175-y
290 REM Draw rectangle
300 PLOT x,y
310 DRAW w,0
320 DRAW 0,h
330 DRAW -w,0
340 DRAW 0,-h
350 NEXT n
360 PAUSE 200
370 NEXT s
380 INK 0: PAPER 7
```

Fig 3.17 Program to draw random rectangles with cut off correction.

If we write a rectangle drawing subroutine it is fairly easy to build in the required tests to avoid out of range errors. Before drawing the first side of the rectangle we simply check to see if x1+w is greater than 255 and if it is, then the value of w is altered to w=255−x1. This reduces the width of the rectangle so that its right-hand edge is at the right-hand screen limit. Before drawing the second side of the rectangle a similar test is carried out on y1+h but here the limit value is 175 and if the rectangle goes off the screen a new value for h is calculated from h=175−y. The top and left sides of the rectangle are simply drawn using the corrected values for w and h. The result on the screen is a rectangle which has been cut off at the screen limit on either its right or top side or perhaps both. To see this working try running the program of Fig. 3.17 which will produce results similar to those shown in Fig. 3.18.

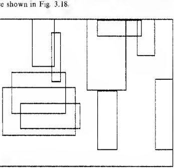


Fig. 3.18. Picture produced by random rectangle program.

Chapter Four
Drawing Techniques

So far we have looked at drawing lines and rectangles but for many purposes we will need to draw more complex figures such as polygons, circles and ellipses. We may also want to draw our figures at an angle to the horizontal. In this chapter we shall explore the techniques involved in doing these things and we'll start by looking at the process of drawing circles. There are in fact several methods which can be used to draw a circle.

Using the CIRCLE command

The easiest way of drawing a circle on the screen is to use the special circle drawing command provided on the Spectrum. This command is called appropriately enough CIRCLE and the command word is obtained by first pressing both SYMBOL SHIFT and CAPS SHIFT keys to get a flashing E cursor and then pressing both the H and SYMBOL SHIFT keys.

The CIRCLE command has the following form:

```
100 CIRCLE x,y,r
```

where x,y are the co-ordinates of the centre of the circle and r is the radius measured in screen units. If we wanted to draw a circle with a radius of 50 units positioned roughly in the centre of the screen then x and y will be 128 and 88 whilst r is 50. Try typing in the direct command

```
CIRCLE 128,88,50
```

and you should get a black circle roughly central on the screen.

To see how the CIRCLE command may be used in a program you might like to try the program listed in Fig. 4.1. Here a series of concentric circles of increasing size is drawn to produce the display

```
100 REM Drawing circles
110 REM using CIRCLE command
120 LET x=128
130 LET y=88
140 FOR r=10 TO 80 STEP 10
150 CIRCLE x,y,r
160 NEXT r
```

Fig. 4.1 Drawing concentric circles using the CIRCLE command.

shown in Fig. 4.2. The values for x and y are the same for all of the circles but r is increased in steps to give successively larger circles.

One important limitation of the CIRCLE command on the Spectrum is that it does not work if any part of the circle goes outside the screen limits. When a part of the circle falls outside the screen limits the computer will simply stop and indicate an integer out of range error.

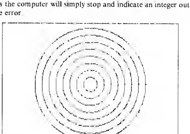


Fig. 4.2 Picture produced by program in Fig. 4.1.

Suppose that we wanted to draw a series of random sized circles all over the screen using the CIRCLE command. To avoid problems we must limit the values of x,y and r for each circle so that the circle does not overlap the screen edges. If we consider x, its minimum value must be r to prevent overlap at the left side and the maximum must be 255−r to prevent overlap at the right side. In the random function we use 255−2r to compensate for the minimum value that we assigned to x, so the final calculation for x is:

x = r + RND*(255−2*r)

A similar technique is used to calculate the value for y and the

```
100 REM Random circle drawing
110 REM using CIRCLE command
120 FOR s=1 TO 10
130 CLS
140 REM Draw screen border
150 PLOT 0,0
160 DRAW 255,0
170 DRAW 0,175
180 DRAW -255,0
190 DRAW 0,-175
200 REM Draw circles
210 FOR n=1 TO 15
220 LET r=5+RND*50
230 LET x=r+RND*(255-2*r)
240 LET y=r+RND*(175-2*r)
250 CIRCLE x,y,r
260 NEXT n
270 PAUSE 200
280 NEXT s
```

Fig 4.3 Program to draw random circles

complete program is shown in fig. 4.3. The result produced on the display will be similar to that shown in Fig. 4.4.

If circles are being drawn and it seems likely that they will overlap the edge of the screen then it is best to use one of the other circle

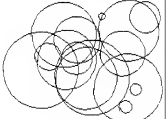


Fig 4.4 Random circles – typical display.

drawing techniques combined with routines which will handle off-screen points either by placing them at the edge of the screen or by providing wraparound so that part of the circle is drawn at the other side of the screen. Let us now look at some other approaches to the drawing of circles using a computer and see how they can be applied on the Spectrum.

The quadratic equation method

The first technique for drawing circles builds up the circle by plotting a large number of dots whose positions are calculated by using one of the mathematical formulae for a circle.

Let us start by taking a small segment of the circle as shown in Fig 4.5. Here point A is at the centre of the circle whilst points B and C are on the circle itself. The lines AB and AC will each have a length

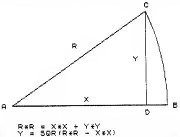


Fig 4.5 Diagram showing derivation of quadratic method for circles

equal to the radius, R, of the circle. In this diagram line AB is horizontal. Let us now drop a vertical line down from point C to meet side AB at point D. This produces a right angled triangle ACD.

To place a dot at each of the points B and C we need to know the X and Y co-ordinates for each of those points and it is convenient to calculate these relative to the centre point of the circle which is point A. The X value for point B is given by the length of side AB, which equals the radius R, and the Y value is 0 since point B is at the same vertical position as point A. When we look at point C the X value is the length of side AD and the Y value is equal to the length of side CD of the triangle ACD.

For a right angled triangle the square of the length of the longest side is the sum of the squares of the lengths of the other two sides. This is our friend Pythagoras's famous theorem.

Let us now use this to calculate the X and Y values for point C. Applying the rule to triangle ACD we get:

(AC)*(AC) = (AD)*(AD) + (CD)*(CD)

or, alternatively, putting in the variables we used for those sides we get:

R * R = (X * X) + (Y * Y)

and this can be rearranged to give us:

Y * Y = (R * R) − (X * X)

from which we can get Y by simply taking the square root. So our calculation for Y now becomes:

Y = SQR((R*R) − (X * X))

The values of X and Y in this equation are measured with reference to the centre point of the circle. Note that for point B the equation for Y is still true since in this case X=R so the term on the right becomes zero and therefore Y=0.

To place the circle at some particular point on the screen we shall have to add in the X and Y co-ordinates for the point where the circle is to be drawn. To avoid confusion we shall call these co-ordinates cx and cy.

In order to plot all of the points around the circle we need to calculate values of y for a series of values of x ranging from −r to +r and the more points we calculate the better the circle will look.

When we take the square root of a number there are in fact two possible answers with the same numerical value, one being positive and the other negative. Thus for each value of X we shall plot a pair of points. To plot the first point of the pair the result of the square root calculation is added to the Y value for the centre of the circle to give a point above the centre line of the circle. The second point has the square root subtracted from the Y value for the centre and the point will lie below the centre line of the circle. Since we are plotting single points and the circumference of the circle is just over 3 times radius R, it is convenient to have a total number of points equal to 4 times R to make up the circle. Remember that we plot two points for each calculation, so a good value for the number of calculation is twice the value of radius R. This is easily achieved by taking all of the X values from X= −R to X = +R. Thus a circle with a radius of 50 screen units would calculate 100 steps and plot a total of 200 points around the circle.

The program shown in Fig. 4.6 draws random sized circles at random centre points on the screen. In this program tests are made for off-screen points and these are corrected to lie at a screen edge to avoid program failure on an out of range error.

```
100 REM Circle by quadratic method
110 LET cx=128
120 LET cy=96
130 LET r=50
140 FOR x=-r TO r
150 LET y=SQR (r*r-x*x)
160 IF cx+x>255 THEN  LET x=255-cx
170 IF cx+x<0 THEN  LET x=0-cx
180 IF cy+y>175 THEN  LET y=175-cy
190 IF cy+y<0 THEN  LET y=0-cy
200 PLOT cx+x,cy+y
210 PLOT cx+x,cy-y
220 NEXT x
```

Fig 4.6 Program to draw circles by quadratic method

With this routine the number of calculations depends upon the size of the circle and it will be seen that the larger circles take a noticeable time to draw. This is because the computer has quite a lot of calculations to carry out. The square root function itself is rather slow in BASIC. If we want to draw circles faster we will need to look at other ways of calculating the points around the circle.

You will note that at the right and left sides of the circle the first few points tend to be rather spread out, especially on the larger radius circles. This can be overcome by plotting more points by increasing X by steps of say 0.5 instead of 1.

The trigonometric method

Instead of plotting a series of dots we can draw a series of short lines which when joined together will form the outline of the circle. For the second method of drawing circles we need to get involved in some simple trigonometry.

Figure 4.7 shows a segment of the circle with two points B and C on the circle itself and point A at the centre of the circle. To draw the segment of the circle we shall draw the line BC.

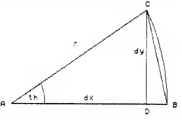


Fig. 4.7. Derivation of trigonometric method for circles

To draw the side BC we need to know the co-ordinates of point C. Let us drop a vertical line from C to point D. The x value for point C is given by the length of line AD and the y value is equal to the length of line CD. This is where the trigonometry comes in.

We will call the angle at point A of the triangle *theta* (θ) which is the name of the Greek letter normally used for labelling angles. In our program we we shall use the variable name 'th' to represent the angle theta.

To find the length of side CD the function we need to use is SIN(theta). The definition of SIN(theta) is that it is the ratio of the length of the side of the triangle opposite angle theta to the length of the hypotenuse (the side opposite the right angle) which is side AC. So in our triangle:

SIN(th) + CD/AC

We already know that AC = radius r. The length of side CD is the change we need to make to y to give the new y value for point C. We shall call this dy. Substituting these new terms in the equation we get:

SIN(th) = dy/r

and if we multiply both sides by r the result becomes:

dy = r * SIN(th)

Having found dy we need to find a value for side AD which is the required change in x and which we shall call dx. Now it just happens that COS(th) is the ratio of the length of the adjacent side (AD) of the triangle to the length of the hypotenuse (AC) so we get:

COS(th) = AD / AC

and substituting the values dx and r gives:

dx = r * COS(th)

To find the co-ordinates for the next point on the circle we apply the same equations but now angle th has a different value.

To draw the circle we must first of all place the cursor at point B for which dx=r and dy=0. This step is carried out by first setting variables x1 = cx+r and y1=cy then using:

```
PLOT x1,y1
```

to plot the first point. The variables cx and cy are the co-ordinates for the centre of the circle.

The next step is to calculate the co-ordinates of point C which are given by the equations:

x2 = cx + dx = cx + r * SIN(th)
y2 = cy + dy = cy + r * COS(th)

and using x2,y2 we can draw the line BC by using:

```
DRAW x2-x1,y2-y1
```

For the next line segment of the circle the values of x1 and y1 are set equal to the values of x2 and y2. The angle th is then increased and new values are calculated for x2,y2 using the new value for the angle th. This process continues until the angle th reaches 360 degrees when a complete circle will have been drawn.

How do we decide on a value for th? Well there are 360 degrees in a complete rotation of the angle th around the circle. To draw a smooth circle the more steps we use the better. A practical value for the number of steps is the number of units of radius r. Suppose we want a circle of radius 60. In this case, each segment of the circle adds 360/60 or 6 degrees to the value of th.

Whilst angles in degrees are familiar to us, the computer doesn't work in degrees but uses radians instead. All we need to know here is that 360 degrees is equal to 2*PI radians so therefore the angle

```
100 REM Circle drawing using the
110 REM trigonometric method
120 BORDER 6
130 FOR s=1 TO 10
140 CLS
150 FOR n=1 TO 10
160 LET r=10+INT (RND*40)
170 LET x=INT (RND*255)
180 LET y=INT (RND*175)
190 GO SUB 500
200 NEXT n
210 PAUSE 100
220 NEXT s
230 STOP
500 REM Circle subroutine
510 LET dt=2*PI/r
520 REM Set starting point
530 LET th=0
540 LET x1=x+INT (r*COS th)
550 LET y1=y+INT (r*SIN th)
560 REM Correct off screen points
570 IF x1>255 THEN  LET x1=255
580 IF x1<0 THEN  LET x1=0
590 IF y1>175 THEN  LET y1=175
600 IF y1<0 THEN  LET y1=0
610 PLOT x1,y1
620 FOR i=0 TO r
630 LET th=th+dt
640 LET x2=x+INT (r*COS th)
650 LET y2=y+INT (r*SIN th)
655 REM Correct off screen points
660 IF x2>255 THEN  LET x2=255
670 IF x2<0 THEN  LET x2=0
680 IF y2>175 THEN  LET y2=175
690 IF y2<0 THEN  LET y2=0
700 DRAW x2-x1,y2-y1
710 LET x1=x2
720 LET y1=y2
730 NEXT i
740 RETURN
```

Fig 4.8 Random circles using the trigonometric method increases by 2*PI/60 radians for each segment of the circle.

The number PI is a constant whose value is approximately 3.14 and it is the ratio of the circumference of a circle to its diameter. We do not need to remember the value for PI because the Spectrum has a special key which allows us to insert PI into a program statement as a constant. To get the term PI into a statement, the CAPS SHIFT and SYMBOL SHIFT keys are pressed together to get extended keyboard mode and then the M key is pressed.

Drawing the circle involves using a simple loop to repeat the calculations and draw a short line segment r times. After each segment of the circle has been drawn the value of th is increased and the values of xl and yl are updated to point to the end of the line that has just been drawn ready for the next drawing step.

To draw our circle we merely calculate a series of values of x and y for values of theta from 0 to 360 degrees (0 to 2*PI radians). The number of points we need depends upon the size of the circle and how accurately we went to draw it. A good figure to us is the number of units of the radius, so for a circle of radius 50 we might use 50 points. The angle theta for each step can be found by simply dividing 2 * PI by the required number of points so the step size would be 2*PI/R. A program for drawing circles using this technique is shown in Fig. 4.8. This program draws a series of random size circles all over the screen to give a result similar to that shown in Fig. 4.9.

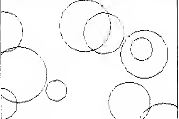


Fig. 4.9 Typical picture from program of Fig. 4.8

The actual circle drawing section is written as a subroutine starting at line 5ØØ. If a number of circles is required it is convenient to use a subroutine for drawing the circle and calling it from the main program whenever a circle is required to be drawn. Sometimes the calculated values for x and y co-ordinates may fall outside the limits of the screen and if used in a DRAW command would produce an error message and stop the program. To avoid this x2 and y2 are tested and if outside the screen limits they are set to the corresponding screen limit value. So if x2<0 then x2 is set to 0. This produces a line at the edge of the screen where part of the circle is outside the screen limit.

The rotation method

A different approach to the calculation of the x,y values for a circle is to base them upon the angle through which the radial line is rotated at each step. In this case the new values for x2 and y2 are calculated from the values for the previous point (x1,y1) rather than from the radius and the total angle.

If we look at Fig. 4.10 the value of y1 is zero so that only the x1 value, which also happens to be equal to r, affects the results. Here we get:

x2 = x1 * COS(th)
y2 = x1 * SIN(th)

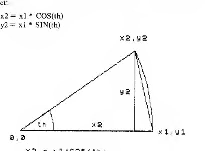


Fig. 4.10 Rotation from the x axis.

Now consider the situation where the radial line is vertical and is moved through angle theta. This is shown in Fig. 4.11. Here the value of x1 is 0 and only the y1 value affects the results. In this case

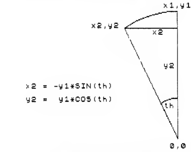


Fig 4.11 Rotation from the y axis

the value of x2 is negative since the point has been shifted to the left of the line where x1=0. Here we get results:

x2 = −y1 * SIN(th)
y2 = y1 * COS(th)

If we combine these two results we can produce a general expression for calculating x2 and y2 for any initial values of x1 and y1. The two new equations are:

x2 = x1 * COS(th) − y1 * SIN(th)
y2 = x1 * SIN(th) + y1 * COS(th)

The big advantage of this approach is that the value of th is constant so we can work out the values of SIN(th) and COS(th) before entering the co-ordinate calculation and line drawing loop, thus eliminating virtually all of the trigonometric calculations which tend to be slow. The program for drawing a circle now becomes as shown in the listing of Fig. 4.12.

```
100 REM Random circles by the
110 REM rotation method
120 FOR n=1 TO 15
130 LET r=10+INT (RND*40)
140 LET x=r+INT (RND*(255-2*r))
150 LET y=r+INT (RND*(175-2*r))
160 GO SUB 500
170 NEXT n
180 STOP
500 REM Circle subroutine
510 LET th=2*PI/r
520 LET x1=r
530 LET y1=0
540 PLOT x+x1,y+y1
550 FOR i=1 TO r
560 LET x2=x1*COS th-y1*SIN th
570 LET y2=x1*SIN th+y1*COS th
580 DRAW x2-x1,y2-y1
590 LET x1=x2
600 LET y1=y2
610 NEXT i
620 RETURN
```

Fig 4.12 Circle drawing by the rotation method

Drawing polygons

If we reduce the number of steps and hence the number of line segments used in the circle drawing routine the result will be a figure with a number of equal straight sides. Such a figure is called a regular *polygon*. If we use the trigonometric method the steps in the angle th become quite large. Suppose we drew an eight-sided polygon, which is called an *octagon*, then the angle the will change by 2*PI/8 at each drawing step. The program listed in Fig. 4.13 will draw a single octagon at the centre of the screen. If we wanted a six sided figure, known as a *hexagon*, then the number of steps in the drawing loop is reduced to 6. Thus the total angle of 2*PI is divided by 6 to give a change in th of 2*PI/6 for each step.

To make a general polygon drawing routine we could introduce a new parameter ns (number of sides) and then modify the program to make the required number of drawing steps. Thus the change in th at each step (dt) will be given by:

dt = 2*PI/ns

```
100 REM Octagon drawing program
110 LET xc=128
120 LET yc=88
130 LET r=50
140 LET dt=2*PI/8
150 LET th=0
160 LET x1=xc+r
170 LET y1=yc
180 PLOT x1,y1
190 FOR n=1 TO 8
200 LET th=dt*n
210 LET x2=xc+r*COS th
220 LET y2=yc+r*SIN th
230 DRAW x2-x1,y2-y1
240 LET x1=x2
250 LET y1=y2
260 NEXT n
```

Fig. 4.13 Program to draw an octagon

To see how this works try running the program shown in Fig. 4.14.

This program will draw a series of figures of increasing size centred on a point near the middle of the screen as shown in Fig.

```
100 REM Nested polygons
110 LET r=12
120 LET xc=128
130 LET yc=88
140 FOR k=3 TO 9
145 REM Set number of sides
150 LET ns=k
160 LET dt=2*PI/ns
170 LET x1=xc+r
180 LET y1=yc
190 PLOT x1,y1
195 REM Draw polygon
200 FOR n=1 TO ns
210 LET th=n*dt
220 LET x2=xc+r*COS th
230 LET y2=yc+r*SIN th
240 DRAW x2-x1,y2-y1
250 LET x1=x2
260 LET y1=y2
270 NEXT n
280 LET r=r+12
290 NEXT k
```

Fig. 4.14 Concentric polygons program

4.15. The figures have an increasing number of sides starting with a triangle.

The program can easily be modified to produce a series of random polygons with different numbers of sides and different sizes. A point to note here is that the values of x, y and r should be chosen so that no point of a polygon falls outside the screen limits. This can be done by using the same technique as in Fig. 4.3.

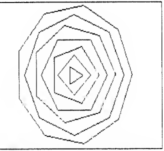


Fig. 4.15. Screen display produced by Fig. 4.14.

Drawing polygons by rotation

Now if we can draw circles using the line rotation technique then it should be possible to draw octagons and polygons as well. For an octagon the angle th will be 2*PI/8 so a subroutine for drawing an octagon would be as shown in Fig. 4.16.

As before we can develop this little routine into a general polygon drawing routine by adding the variable k (number of sides) and we can produce a program which draws polygons with from 3 to 12 sides in various sizes all over the screen. This is shown in Fig. 4.17.

This procedure for drawing polygons can be used as a general method in any program. Note that it will draw squares and triangles but the triangles will always be equal sided ones.

```
100 REM Octagon by rotation method
110 LET xc=128
120 LET yc=88
130 LET r=50
140 GO SUB 500
150 STOP
500 REM Octagon drawing subroutine
510 LET x1=r
520 LET y1=0
530 LET th=2*PI/8
540 LET sn=SIN th
550 LET cn=COS th
560 PLOT xc+x1,yc+y1
570 FOR n=1 TO 8
580 LET x2=x1*cn-y1*sn
590 LET y2=x1*sn+y1*cn
600 DRAW x2-x1,y2-y1
610 LET x1=x2
620 LET y1=y2
630 NEXT n
640 RETURN
```

Fig. 4.16 Drawing an octagon by rotation method

```
100 REM Random polygons
110 FOR j=1 TO 20
120 LET r=10+INT (RND*40)
130 LET xc=r+INT (RND*(255-2*r))
140 LET yc=r+INT (RND*(175-2*r))
150 LET k=3+INT (RND*5)
160 GO SUB 500
170 NEXT j
180 STOP
490 REM Polygon subroutine
500 LET th=2*PI/k
510 LET sn=SIN th
520 LET cn=COS th
530 LET dx=r
540 LET dy=0
550 PLOT xc+dx,yc+dy
560 FOR n=1 TO k
570 LET xr=dx*cn-dy*sn
580 LET yr=dx*sn+dy*cn
590 DRAW xr-dx,yr-dy
600 LET dx=xr
610 LET dy=yr
620 NEXT n
630 RETURN
```

Fig. 4.17 Random polygons program

Star shaped figures and wheels

The polygon drawing routine can easily be modified to draw star shaped figures. In this case a line is drawn from the centre to each calculated point of the polygon by changing the drawing procedure. The program listing in Fig. 4.18 will draw a pattern of random star shaped figures on the screen.

In this case the line is drawn from each corner of the polygon back to the centre so that the lines radiate from the centre like the spokes of a wheel.

Wheel shapes can be drawn by firstly drawing the star and then drawing a circle of the same radius around the same centre point.

```
100 REM Star shaped figures
110 FOR j=1 TO 20
120 LET r=10+INT (RND*40)
130 LET xc=r+INT (RND*(255-2*r))
140 LET yc=r+INT (RND*(175-2*r))
150 LET k=3+INT (RND*5)
160 GO SUB 500
170 NEXT j
180 STOP
490 REM Star shape subroutine
500 LET th=2*PI/k
510 LET sn=SIN th
520 LET cn=COS th
530 LET dx=r
540 LET dy=0
550 FOR n=1 TO k
560 LET xr=dx*cn-dy*sn
570 LET yr=dx*sn+dy*cn
580 PLOT xc,yc
590 DRAW xr,yr
600 LET dx=xr
610 LET dy=yr
620 NEXT n
630 RETURN
```

Fig 4.18 Program to draw star shape figures

Scaling and stretching

In drawing squares, polygons and circles the size of the displayed figure depends upon the value of W or R that we use in the drawing routine. Thus by altering W or R we can alter the size or scale of the figure.

In the case of the rectangle there are two scaling figures, one for width (W) and one for height (H). In effect we have a square which has been stretched or compressed in one direction. Assuming that we apply stretching only horizontally or vertically this just means that the x and y scale values are different.

We could apply the stretching idea to other figures by putting in two extra variables which would be the x and y scale factors. To achieve the correct results the reference point around which the figure is drawn should be at the centre of the figure. For polygons and circles this is always true in the drawing methods we have used. In this case the scale factors are used as multipliers for the dx and dy terms in the drawing calculations. Note that the scale factors are not applied to the screen co-ordinates cx,cy around which the figure is drawn.

Let us consider a circle and we will use the trigonometric drawing method as shown in Fig. 4.19. Two new terms sx and sy are now used and the values of dx and dy are multiplied by sx and sy respectively before the figure is drawn. First the circle is drawn at the left of the screen with sy being increased in steps from 0 to 1 and sx constant at 1. Next the circle is drawn at the right of the screen with sx increasing from 0 to 1 and sy set at 1. Finally both sx and sy are varied from 0 to 1.

If sx and sy are both 1 then the figure drawn will be a circle. If sx<1 and sx<=1 the figure becomes an ellipse with the longer axis horizontal. If sx>1 and sy>=1 the ellipse will have its long axis vertical. If sx or sy is negative this will simply have the effect that the figure is drawn backwards. If we had a figure that was not symmetrical then the left side would be displayed at the right or the top would move to the bottom giving a mirror image effect.

```
100 REM Scaling and stretching
110 REM applied to a circle
120 LET xc=40
130 LET yc=88
140 LET r=40
145 REM Scaling applied to y
150 FOR a=1 TO 0 STEP -.2
160 LET sx=1
170 LET sy=a
180 GO SUB 500
190 NEXT a
200 LET xc=210
205 REM Scaling applied to x
210 FOR a=1 TO 0 STEP -.2
220 LET sx=a
230 LET sy=1
240 GO SUB 500
250 NEXT a
260 LET xc=128
265 REM Scaling of y then x
270 FOR a=1 TO 0 STEP -.2
280 LET sx=1
290 LET sy=a
300 GO SUB 500
310 NEXT a
320 FOR a=1 TO 0 STEP -.2
330 LET sx=a
340 LET sy=1
350 GO SUB 500
360 NEXT a
370 STOP
490 REM Circle subroutine
500 LET dt=2*PI/r
510 LET x1=xc+sx*r
520 LET y1=yc
530 PLOT x1,y1
540 FOR n=1 TO r
550 LET x2=xc+sx*r*COS (n*dt)
560 LET y2=yc+sy*r*SIN (n*dt)
570 DRAW x2-x1,y2-y1
580 LET x1=x2
590 LET y1=y2
600 NEXT n
610 RETURN
```

Fig 4 19 Demonstration of ellipse drawing

Rotation of figures

The figures we have produced so far have all been drawn with their x axis horizontal. Suppose, however, we want to draw a rectangle but have it displayed tilted at an angle as shown in Fig. 4.20. We have already seen that a point can readily be rotated relative to another

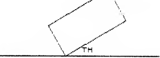


Fig. 4.20. Diagram showing a shape rotated by angle TH.

point on the screen and this technique was used for drawing polygons and circles. If we can rotate one point then we can just as easily rotate all of the points in a figure. In this case the rotation equations are applied to each point on the figure in turn to calculate a new point position for the rotated figure. When the figure is drawn using the new set of points it will be tilted relative to the horizontal.

To find the rotated values for DX and DY we use:

XR = DX*COS(TH) - DY*SIN(TH)

and

YR = DX*SIN(TH) + DY*COS(TH)

where DX and DY are the co-ordinates of each point measured relative to the point about which we want to rotate the figure. The angle of rotation is TH radians relative to the positive x axis.

Let us start with a rectangle and assume that we are using the bottom left-hand corner as a reference point about which the rectangle will be rotated. We shall assume that the rectangle has width W and height H and that the angle through which it is to be rotated is TH.

We can start by setting XI,YI to the co-ordinates of the reference point at the bottom left corner which has actual screen co-ordinates XC,YC. The first line to be drawn is the bottom side so the first point to be rotated is at the right bottom corner. For this point the X and Y offsets DX and DY, measured from the bottom left corner which has actual screen co-ordinates XC,YC are DX = W and DY = 0. At this point we can calculate the rotated position of this point (XR, YR) by substituting the values for DX and DY in the rotation equations.

To draw the first line we start by setting X1 and Y1 to XC and YC respectively and then plotting a point. Next we calculate the co-ordinates of the other end of the line X2,Y2 by adding the values for XR and YR to the reference co-ordinates XC,YC as follows:

```
X2 = XC=YR
Y2 = YC+YR
```

The first line can be drawn by using the X1,Y1 and X2,Y2 co-ordinates for each end of the line in a DRAW statement as follows:

```
DRAW X2-X1,Y2-Y1
```

To draw the second side of the rectangle we now set X1,Y1 equal to the values X2,Y2 so that the new line starts from the end of the first. The DX value for the second point is still equal to W, but the DY value is now equal to height H. With these new values we can again apply the rotation equations to obtain new values for XR and YR and for X2 and Y2. Now the second side can be drawn. This process is then repeated for the remaining two sides. Since the rotation and drawing steps are repeated it is convenient to make these into a subroutine and the program for drawing a simple rotated rectangle would be as shown in Fig. 4.21.

The rotation effect when combined with scale changes can produce interesting spiral patterns as shown by the program listed in Fig. 4.22.

We can apply this technique of rotation to any of the figures that can be generated by a series of mathematical steps. When we have an irregular figure however things are a little more difficult. For such figures it is best to make up a table of data points for each of the corners of the figure. These X,Y points are all measured relative to some point on the figure about which it is to be rotated. This may be the centre of the figure or it may be point on the outside of the figure. To draw the figure we simply take the points in turn and draw lines linking them. One further problem arises however. Sometimes we may just want to move to the next point without drawing a line. This can be catered for by producing a third data array which we shall call L. If L is set at 1 a line is drawn and if L is set at 0 no line is drawn.

Having produced the table of X,Y and L values we can now proceed to draw the figure using much the same technique as we did for drawing the rectangle, except that now the X and Y values

```
100 REM Rotated rectangle
110 LET xc=128
120 LET yc=40
130 LET w=100
140 LET h=30
150 LET dt=PI/6
160 LET th=0
170 FOR n=1 TO 6
180 LET x1=xc
190 LET y1=yc
200 PLOT x1,y1
210 LET x2=xc+w*COS th
220 LET y2=yc+w*SIN th
230 DRAW x2-x1,y2-y1
240 LET x1=x2
250 LET y1=y2
260 LET x2=xc+w*COS th-h*SIN th
270 LET y2=yc+w*SIN th+h*COS th
280 DRAW x2-x1,y2-y1
290 LET x1=x2
300 LET y1=y2
310 LET x2=xc-h*SIN th
320 LET y2=yc+h*COS th
330 DRAW x2-x1,y2-y1
340 LET x1=x2
350 LET y1=y2
360 LET x2=xc
370 LET y2=yc
380 DRAW x2-x1,y2-y1
390 LET th=th+dt
400 NEXT n
```

Fig 4.21 Program to draw a series of rotated rectangles

are taken successively from the arrays. The value for X1,Y1 is initially set at the screen co-ordinates about which we want to draw the shape and then X2,Y2 are calculated using the rotation equations and adding the rotated values to XC and YC respectively. Before the line is drawn L is checked and if it is 0 the DRAW command is skipped. If no line is drawn the PLOT command for the start point of the next line will move the graphics cursor into position ready to draw the new line. After each line has been processed the values of X1,Y1 are updated to the values of

```
100 REM Patterns using rotation
110 REM and scaling
120 LET th=0
130 LET dt=PI/12
140 LET xc=128
150 LET yc=88
160 FOR w=2 TO 70 STEP 2
170 LET x1=xc
180 LET y1=yc
185 PLOT x1,y1
190 LET dx=w
200 LET dy=0
210 GO SUB 500
220 LET x1=x2
230 LET y1=y2
240 LET dx=w
250 LET dy=w
260 GO SUB 500
270 LET x1=x2
280 LET y1=y2
290 LET dx=0
300 LET dy=w
310 GO SUB 500
320 LET x1=x2
330 LET y1=y2
340 LET dx=0
350 LET dy=0
360 GO SUB 500
370 LET th=th+dt
380 NEXT w
390 STOP
490 REM Rotation subroutine
500 LET x2=xc+dx*COS th-dy*SIN th
510 LET y2=yc+dx*SIN th+dy*COS th
520 DRAW x2-x1,y2-y1
530 RETURN
```

Fig 4 22 Patterns using rotation and scaling

X2,Y2 ready for the next line to be dealt with This process continues until the figure is complete

If there are several points in the figure this rotation routine can be speeded up somewhat. Since the angle TH is constant for all points in the figure, the values of SIN(TH) and COS(TH) could be calculated before starting the drawing loop. Now the results of these calculations SN and CN can be used inside the loop thus saving many trigonometric calculations which tend to slow down program execution. The program would then become as shown in Fig. 4.23.

Here, since we have used X and Y arrays to define the points in the figure, the variables XC and YC have been used to define the origin point around which the figure will be drawn on the screen.

```
100 REM Rotating an irregular shape
110 DIM x(5)
120 DIM y(5)
130 DIM l(5)
135 REM Set up data for figure
140 FOR n=1 TO 5
150 READ x(n),y(n),l(n)
160 NEXT n
170 DATA 20,0,0
180 DATA 60,0,1
190 DATA 60,-15,0
200 DATA 40,0,1
210 DATA 60,15,1
220 LET xc=128
230 LET yc=88
240 LET dt=2*PI/10
250 LET th=0
260 FOR f=1 TO 10
270 LET sn=SIN th
280 LET cn=COS th
290 LET x1=xc
300 LET y1=yc
310 GO SUB 500
320 LET th=th+dt
330 NEXT f
340 STOP
490 REM Figure drawing subroutine
500 PLOT x1,y1
510 FOR n=1 TO 5
520 LET x2=xc+x(n)*cn-y(n)*sn
530 LET y2=yc+x(n)*sn+y(n)*cn
540 IF l(n)=1 THEN  DRAW x2-x1,y2-y1
550 PLOT x2,y2
560 LET x1=x2
570 LET y1=y2
580 NEXT n
590 RETURN
```

Fig. 4.23 Rotation of an irregular shape.

Chapter Five
New Characters and Shapes

So far when printing pictures on to the screen we have used the normal text character set and the mosaic graphics symbol set to produce displays. For most applications this will be perfectly adequate but sometimes there will be situations where we may want to print a symbol that is not provided in the standard symbol set. For example, we might want to produce the symbols representing the suits of playing cards or, perhaps, for a program dealing with mathematics we might want to use symbols from the Greek alphabet.

One possible solution to the problem might be to actually draw the required symbol using the high resolution PLOT and DRAW commands. This would involve drawing the symbol on a piece of paper and then working out the required sequence of drawing steps needed to produce the shape on the screen. If only one symbol or pattern is required and the shape is easy to draw then this approach is reasonably practical although rather tedious. When several different symbols are needed and particularly where the symbols are to be mixed with normal printed text, as in the case of Greek or Russian letters, the drawing method becomes impractical. What we really need is a method for producing special symbols that can be printed in the same way as other text symbols.

The Spectrum does in fact have a facility by which we can produce a set of custom designed symbols and use them in the same way as the standard character set. We shall now take a look at how this works.

The user defined characters

If we print out all of the available characters on the Spectrum using the program shown at the beginning of Chapter Two, it will be found that after the mosaic graphics symbols there are the letters A to U. It may seem rather odd that the patterns for these twenty symbols are duplicated. In fact these are the user defined symbols in which the pattern of dots can be set up by the user to give any desired symbol. By reprogramming the dot patterns for these symbols we can generate Greek or Russian letters or perhaps even the Japanese Katakana symbols and Chinese characters. Apart from text symbols we can also program the dot patterns of these symbols to display space invaders, rockets, playing card suit symbols and so on.

Unlike the normal text and mosaic graphics symbols which have their dot patterns stored in a Read Only Memory (ROM), the user defined symbols have their dot patterns held in part of the normal read write memory. When the Spectrum is switched on it automatically copies the dot patterns for the letters A to U into the memory locations reserved for the user defined symbol dot patterns. If this were not done the symbols would just be random patterns of dots. The memory area used for storing the custom symbol patterns is at the top of the main memory. The actual memory addresses used will depend upon whether the Spectrum has a 16K or 48K memory.

Like the other text characters, each of the user defined symbols has 8 rows with 8 dots in each row and each dot may be either on or off. In the computer each memory word has 8 bits, each of which may be set as a 1 (on) or a 0 (off) so it is convenient to store one row of dots from the character pattern into one memory word. The 8 rows of dots making up the character are then stored in 8 successive memory words. If we want to create a new symbol then the new dot pattern must be written into a set of eight memory locations in the user defined graphics area of memory.

Programming a new symbol

The first step in creating a new symbol is to work out the dot pattern that is needed to build up the symbol. This can be done by simply drawing a grid with eight rows of squares and eight squares in each row as shown in Fig. 5.1. Squares are then shaded in to pick out the shape of the desired symbol.

Once the dot pattern has been worked out the next step is to calculate the numbers that have to be stored in the memory. Figure 5.1 shows the layout of a typical user defined graphics character. In fact this is the same basic pattern of a little man that we set up earlier using the mosaic graphics symbols. In the diagram the black dots are

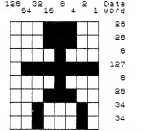


Fig 5 1 Conversion of dot patterns to numbers.

in the INK colour and will be represented by '1's in the computer word. Each data bit in the word has a numerical value starting with 1 for the right-hand end bit and working up in the sequence 2, 4, 8, 16 and so on for successive bits as we move to the left through the data word. The actual value of each bit is shown at the top of the diagram.

When a dot is in INK colour the corresponding data bit in the word is set at 1 but if the dot is in PAPER colour the bit is set at 0. To find the actual decimal number that has to be fed into the computer we can simply add together the numerical values for all of the bits in the word that are set at '1'. This gives a number in the range 0 to 255.

To set up the dot pattern in memory we now have to write the sequence of eight numbers into eight successive memory locations and this can be done by using the POKE command which takes the form

POKE address, value

where address is the actual address in the computer memory and value is the number that we want to write into that address.

All we have to do now is put the data words into the right place in memory. Fortunately we do not need to know the actual address as a number because the computer can find that itself. Suppose we want to put our dot pattern into the first available user defined symbol space. You will remember that this initially displayed an A symbol and to put in the data for the first row we can use

POKE USR"a",data

where data is the value for the top row of dots. To set up the next row of dots we can use:

POKE USR"a"+1,data

where data is now the number for the second row of dots. We can continue in the same way for the remaining rows of dots.

Having set up the symbol pattern, how can it be displayed on the screen? We can use a PRINT statement with a CHR$ term as we did earlier for mosaic graphics symbols. The character codes for the user defined symbols actually run from 144 to 164. If we chose USR"a" to set the POKE address then the character code will be 144. In Fig. 5.2 the USR address codes and the associated character code for the dot patterns they select, are listed.

USR letter	Custom symbol Character code	USR letter	Custom symbol Character code
A	144	L	155
B	145	M	156
C	146	N	157
D	147	O	158
E	148	P	159
F	149	Q	160
G	150	R	161
H	151	S	162
I	152	T	163
J	153	U	164
K	154		

Fig. 5.2 The USR address codes and corresponding user defined character symbol codes.

Another way of printing the special symbols is to use the graphics shift mode of the keyboard. This is entered by pressing the CAPS SHIFT and 9 keys together which changes the cursor to a flashing G. For mosaic symbols the keys with those symbols on would be used but for the user defined symbols we simply have to press one of the letter keys A to U according to which special symbol we want to print.

There is another way in which we can define the dot patterns for the symbol. This actually makes use of the binary word consisting of a string of '1's and '0's. To let the computer know that the data is in binary form, we put the instruction BIN in front of the string of binary data bits. Thus the top row of our little man figure could be written as:

BIN ØØØ111ØØ

Note that BIN is an instruction word and is obtained by pressing the B key when the keyboard is operating in the extended mode (flashing E cursor).

The program listed in Fig. 5.3 makes use of this method of specifying the dot pattern.

```
100 REM Setting up special
110 REM graphics symbol using BIN
120 POKE USR "a",BIN 00011100
130 POKE USR "a"+1,BIN 00011100
140 POKE USR "a"+2,BIN 00001000
150 POKE USR "a"+3,BIN 01111111
160 POKE USR "a"+4,BIN 00001000
170 POKE USR "a"+5,BIN 00011100
180 POKE USR "a"+6,BIN 00100010
190 POKE USR "a"+7,BIN 00100010
200 FOR n=1 TO 20
210 LET r=INT (RND*20)
220 LET c=INT (RND*30)
230 PRINT AT r,c;CHR$ 144;
240 NEXT n
```

Fig. 5.3. Using the BIN command to set up dot patterns

More complex patterns

Sometimes we may find that the shape we want to produce for our symbol is too complex to be displayed on an 8 × 8 array of dots. This can be easily overcome by making up the desired pattern from a group of special symbols and printing the groups alongside one another. This follows the same principle that we used in Chapter Two when we built up the little man figure from a pattern of 4 × 4 mosaic graphics symbols.

An alternative approach is shown in the program listed in Fig. 5.4. Here a two-dimensional array d has been set up with 8 rows and 8 columns. Each number in the array is set to either 1 or Ø according to whether a dot is required at that position in the character dot pattern.

```
100 REM Setting up dot array for
110 REM symbol or sprite
120 DIM d(8,8)
125 REM Set up dot array
130 FOR b=1 TO 8
140 EOR a=1 TO 8
150 READ d(a,b)
160 NEXT a
170 NEXT b
180 DATA 0,0,0,1,1,1,0,0
190 DATA 0,0,0,1,1,1,0,0
200 DATA 0,0,0,0,1,0,0,0
210 DATA 0,1,1,1,1,1,1,1
220 DATA 0,0,0,0,1,0,0,0
230 DATA 0,0,0,1,1,1,0,0
240 DATA 0,0,1,0,0,0,1,0
250 DATA 0,0,1,0,0,0,1,0
260 FOR n=1 TO 20
270 LET x=INT (RND*200)
280 LET y=10+INT (RND*150)
285 REM Plot symbol
290 FOR b=0 TO 7
300 FOR a=0 TO 7
310 IF d(a+1,b+1)=0 THEN  GO TO 330
320 PLOT x+a,y-b
330 NEXT a
340 NEXT b
350 NEXT n
```

Fig. 5.4. Using an array to store a dot pattern for a sprite.

To display the symbol two counting loops are used to scan through all of the values of d. At the same time the graphics cursor position is moved over the screen to scan out the dot pattern. At each step the value of d is tested to see if it is a '1' or a '0'. Where d is a '1' a dot is plotted on the screen but where d is a '0' the plot instruction is skipped and the program moves on to process the next dot location.

This technique of producing shapes uses rather a lot of memory, but is possibly easier to set up and more flexible than using an array of special symbols to build up the shape.

The POINT command

Sometimes we may want to know the state of a particular point on the screen and this can be done quite easily by using the POINT command. The POINT command word is obtained by selecting the extended keyboard mode where the flashing E cursor appears and then using the 8 key with SYMBOL SHIFT.

The complete POINT command may take the form:

```
1ØØ LET p = POINT (x,y)
```

where x and y are the position co-ordinates for the point we wish to check. If the point on the screen is set to the INK colour then the resultant value for p will be 1, whereas if the point is in PAPER colour then p will be Ø.

Although POINT will tell us whether a selected point on the screen is 'on' or 'off', it will not tell us the actual colour of the point and to find this out we would need to find the colour attributes of the character space that contains the point we are looking at. This can be done by using the ATTR command which we shall examine in Chapter Six.

The POINT command can also be used in an IF statement as follows:

```
1ØØ IF POINT (x,y)=1 THEN GOTO 2Ø
```

Here if the point is turned on the result of the IF test is 'true' and the program will jump to line 2ØØ. If the point is turned off then the program continues with the next statement since the result of the IF operation will be false. If you want the program to jump when the dot is turned off then the IF statement should be changed to:

```
1ØØ IF POINT (x,y) = Ø THEN GOTO 2ØØ
```

We will now use the POINT command to produce some rather interesting manipulation of the dot patterns of displayed symbols.

Positioning symbols on the high resolution screen

The easiest way of inserting text into a high resolution display is to make use of the PRINT AT statement by which the symbol is printed directly on to the screen at a specified point. This technique is quite adequate for many purposes but it is limited to placing symbols into the normal symbol positions on the screen. There will, however, be some occasions when we want to place a symbol at a specific point on the screen which may lie between the normal print positions. This can be achieved fairly easily and we shall now examine the way in which this is done

Remember that a symbol simply consists of an array of dots and if we use the PLOT command dots can readily be set up at any point on the screen. Let us suppose we want to take the symbol A and place it at some random point on the screen. The first thing we need to know is the pattern of dots that make up the A symbol. The easiest way to get the dot pattern is to print the A at position ∅,∅ on the screen. We now know the exact position of this pattern of dots and we can use the POINT command to discover which dots are in INK colour and which are PAPER colour. By using a simple loop operation we can examine each dot of the printed symbol at a time and then we can print a copy of it at any desired point on the screen. The program listed in Fig. 5.5 shows how the A symbol can be printed at a point near the centre of the screen using this technique.

```
100 REM Placing a symbol at the
110 REM screen centre by dot copy
120 PRINT AT 0,0;"A";
130 LET x=128
140 LET y=88
150 REM Copy dot pattern
160 FOR b=0 TO 7
170 FOR a=0 TO 7
180 IF POINT (a,175-b)=0 THEN  GO TO 200
190 PLOT x+a,y-b
200 NEXT a
210 NEXT b
```

Fig 5.5 Transfer of a symbol by dot copying.

The required symbol is first printed at the top left corner of the screen to give the dot pattern that we are going to copy. The upper row of dots in this pattern have x locations running from ∅ to 7 and their y position is 175. The next row of dots also have the same x positions but y is now 174 and successive rows are the same except that y is reduced by 1 for each row. To scan the dots we can set up two count loops with variables a and b which both run from ∅ to 7.

For the POINT command the x and y parameters will be a and 175−b and we simply have to check if the result is 1 or not to see if the dot is in INK colour. Having examined the dot pattern we now have to plot a copy of the dot pattern at some other desired point on the screen. First we must define the point at which we want to plot the symbol and here it is convenient to select the x,y co-ordinates as the point where the top left corner of the displayed symbol is to be. Now to plot the points we simply use PLOT x+a,y−b where a and b are

```
100 REM Placing a symbol at random
110 REM positions by dot copying
120 PRINT AT 0,0;"A";
130 FOR n=1 TO 50
140 LET x=8+INT (RND*240)
150 LET y=8+INT (RND*160)
160 GO SUB 400
170 NEXT n
180 STOP
390 REM Copy symbol to point x,y
400 FOR b=0 TO 7
410 FOR a=0 TO 7
420 IF POINT (a,175-b)=0 THEN  GO TO 440
430 PLOT x+a,y-b
440 NEXT a
450 NEXT b
460 RETURN
```

Fig. 5.6 Transferring symbols to random screen positions

the same count values as we used in the POINT command. If the POINT test shows a lit dot then a dot is plotted in the new symbol position, but if the dot is off, the PLOT instruction is skipped.

As the two loops progress the dot pattern will be copied from the top left corner to the new position. Using this routine it is possible to plot two symbols overlapping quite easily. This is demonstrated in the program listed in Fig. 5.6 where the symbol is copied to random

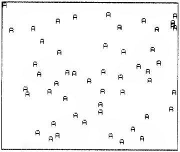


Fig. 5.7 Typical random position symbol display

positions around the screen. This produces a screen display similar to that shown in Fig. 5.7. Note here that we can now have symbols actually overlapping one another. This method of setting up text symbols on the screen is particularly useful when text has to be inserted into a high resolution graphics drawing.

Rotating the symbols

By slightly changing the copying loop we can now produce some more interesting effects. Suppose we want to write the symbol upside down on the screen. If we change the y term of the PLOT command to y+b this effectively inverts the symbol, since the top row of dots that was read from the master pattern now becomes the bottom row of dots in the plotted symbol and the sequence of all other rows also changes. An important point to note here is that the plotted symbol will now have its lower left corner at the specified x,y position. The position x,y must be chosen so that the y co-ordinate of the top row of the plotted symbol does not go beyond 175, otherwise the program will stop on an error.

If you want the symbol upside down but with its top left corner still at x,y then the PLOT command must be changed to:

```
PLOT x+a,y-8+b
```

This would allow the symbol to be inserted easily into a row of printed text symbols.

To turn a symbol back to front so that it looks like a mirror image, we can apply a similar process to the x term of the PLOT command so that it becomes:

```
PLOT x+8-a,y-b
```

Now let us get a little more adventurous and see what happens if we swap the offset terms a and b in the PLOT command to give:

```
PLOT x+b,y+a
```

This produces a symbol which has been turned on its side since the horizontal rows of the original pattern have now been plotted vertically. To turn the symbol over on to its other side we simply have to change the command to:

```
PLOT x-b,y+a
```

```
100 REM Rotation of text symbols
110 REM using dot copying
120 PRINT AT 0,0;"A";
130 LET x=124
140 LET y=96
150 GO SUB 400
160 LET x=124
170 LET y=64
180 GO SUB 500
190 LET x=112
200 LET y=76
210 GO SUB 600
220 LET x=144
230 LET y=84
240 GO SUB 700
250 PRINT AT 0,0;" ";
260 STOP
390 REM Normal symbol
400 FOR b=0 TO 7
410 FOR a=0 TO 7
420 IF POINT (a,175-b)=0 THEN  GO TO 440
430 PLOT x+a,y-b
440 NEXT a
450 NEXT b
460 RETURN
490 REM Inverted symbol
500 FOR b=0 TO 7
510 FOR a=0 TO 7
520 IF POINT (a,175-b)=0 THEN  GO TO 540
530 PLOT x+a,y+b
540 NEXT a
550 NEXT b
560 RETURN
590 REM Symbol rotated to left
600 FOR b=0 TO 7
610 FOR a=0 TO 7
620 IF POINT (a,175-b)=0 THEN  GO TO 640
630 PLOT x+b,y+a
640 NEXT a
650 NEXT b
660 RETURN
690 REM Symbol rotated to right
```

```
700 FOR b=0 TO 7
710 FOR a=0 TO 7
720 IF POINT (a,175-b)=0 THEN  GO TO 740
730 PLOT x-b,y-a
740 NEXT a
750 NEXT b
760 RETURN
```

Fig 5.8 Rotation of symbols by dot copying

Once again in these commands a correction may be made to the basic x,y values so that the top left corner of the new symbol will still be positioned at point x,y on the screen.

The program listed in Fig 5.8 draws symbols in all four orientations and will produce a display similar to that shown in Fig 5.9.

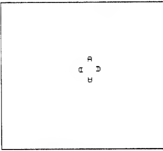


Fig 5.9 Display of rotated symbols

Bigger and better characters

Suppose we want to produce a double width symbol on the screen. To get double width we can simply use a second PLOT command to place a dot alongside the first. Of course we must also have two blank spaces for each blank dot as well. This can readily be achieved by using an offset of twice a so that the plot action moves two points

```
100 REM Producing large symbols
110 LET cy=165
115 REM v=symbol height
120 FOR v=1 TO 5
130 LET cx=10
140 LET cy=cy-8*v
145 REM h=symbol width
150 FOR h=1 TO 5
160 PRINT AT 0,0;"$";
170 LET cx=cx+10*h
180 GO SUB 500
190 NEXT h
200 NEXT v
220 PRINT AT 0,0;" ";
230 STOP
490 REM Symbol copying subroutine
500 FOR y=0 TO 7
510 FOR x=0 TO 7
520 IF POINT (x,175-y)=0 THEN  GO TO 580
530 FOR k=0 TO v-1
540 FOR j=0 TO h-1
550 PLOT cx+h*x+j,cy-v*y+k
560 NEXT j
570 NEXT k
580 NEXT x
590 NEXT y
600 RETURN
```

Fig. 5.10 Program to produce bigger symbols

for each point in the original pattern.

It is equally easy to generate double height symbols. In this case the doubling up process is applied to the b offset instead of to the a offset. Now each row of dots in the original pattern is scanned twice and produces two rows of dots one above the other in the plotted copy character.

By combining the two actions we can produce double size symbols. Further development along these lines will allow treble or quadruple size symbols to be produced from the original dot pattern. An important point to watch when using any of the dot pattern copying routines is to make sure that none of the dots are allowed to go beyond the screen limits.

The program listed in Fig. 5.10 gives some idea of the possibilities of this technique and shows a range of symbols in sizes up to five times as large as the original character. The display produced on the

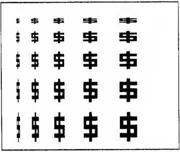


Fig 5 11 Big symbol display

screen by this program is shown in Fig. 5 11 This technique can be very useful for providing bold titles on the screen display.

Sloping Symbols

Sometimes we may want to produce italic style symbols where the displayed symbol is inclined at an angle instead of being vertical. This result can be obtained by subtracting the b offset from the combined a and x term. Now each successive row of dots in the symbol is displaced one position to the left so that the symbol is drawn at an angle and looks very much like a rather exaggerated italic symbol. A more realistic looking italic symbol is produced by using INT (b/2) instead of b, since this reduces the slope of the symbol.

If instead of subtracting the b term it is added to the x term then the symbols will slope in the opposite direction. The results of this type of operation can be seen by running the program listed in Fig. 5 12. The results produced on the screen are shown in Fig. 5 13 which demonstrates the effect produced on a variety of symbol sizes and shapes.

Other possibilities with which you might experiment are to add the a offset to the y term, which will give a character with sloping

```
100 REM Large italic symbols
110 LET cy=165
115 REM Set symbol height (v)
120 FOR v=1 TO 5
130 LET cx=10
140 LET cy=cy-8*v
145 REM Set symbol width (h)
150 FOR h=1 TO 5
160 PRINT AT 0,0;"$";
170 LET cx=cx+10*h
180 GO SUB 500
190 NEXT h
200 NEXT v
210 PRINT AT 0,0;" ";
220 STOP
490 REM Symbol copy subroutine
500 FOR y=0 TO 7
510 FOR x=0 TO 7
520 IF POINT (x,175-y)=0 THEN  GO TO 580
530 FOR k=0 TO v-1
540 FOR j=0 TO h-1
550 PLOT cx+h*x+j-y*h/2,cy-v*y+k
560 NEXT j
570 NEXT k
580 NEXT x
590 NEXT y
600 RETURN
```

Fig 5.12 Creating italic style symbols

horizontal lines, or to combine both operations, which will draw the character rotated through 45 degrees. Since this rotation technique does not follow the normal rotation equations, the shape of the symbol will be distorted when it is rotated in this way. I will leave you to experiment with the various possible combinations that can be applied to the plotting of the copied character.

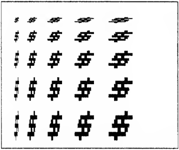


Fig. 5.13 Display of italic symbols

Chapter Six
More About Colour

So far in this book we have used the INK and PAPER commands to set up the drawing or foreground colour (INK) and the background colour (PAPER). There are several other commands which are concerned with the colour on the screen and we shall now explore these to see how we can obtain many more than the eight standard colours of the INK and PAPER commands.

Bright and flashing colours

Sometimes we may wish to emphasise parts of a text display to attract the viewer's attention. Examples of this might be to warn of some potentially dangerous situation or to indicate that some action is required. An example of this is the flashing cursor on the Spectrum text display which shows the position of the text cursor and indicates where the next symbol printed to the display will be located.

One way of making part of the text stand out is to use the command BRIGHT. This command can have either a 1 or a 0 following it. The command BRIGHT 1 causes any new symbols printed on the screen to be brighter than normal and it will also cause the background colour for those symbols to be brighter. The BRIGHT command has no effect on black symbols or backgrounds. To restore the display to normal use the BRIGHT 0 command. Note that like the INK command, BRIGHT only affects symbols which are printed after the BRIGHT 1 command has been used.

Sometimes, depending upon the particular television receiver used for the display, the use of BRIGHT 1 will cause the overall brightness of the picture to fall.

Another way of drawing attention to a particular part of the screen is to use the FLASH command. When FLASH 1 is used any new characters printed will flash on and off. In fact what happens is that the foreground and background colours in the character space alternate so that at one moment there might be a black symbol on a white background and this alternates with a white symbol on a black background. As with the BRIGHT command if we insert a FLASH 0 command the following symbols will be displayed in the normal

```
100 REM Bright and flashing symbols
110 FOR k=1 TO 4
120 PRINT AT k,0;" ";
130 FOR j=0 TO 7
140 INK j
150 PRINT CHR$ (136+j);
160 PRINT CHR$ (136+j);
170 PRINT CHR$ (136+j);
180 NEXT j
190 NEXT k
200 FOR k=5 TO 8
210 BRIGHT 1
220 PRINT AT k,0;" ";
230 FOR j=0 TO 7
240 INK j
250 PRINT CHR$ (136+j);
260 PRINT CHR$ (136+j);
270 PRINT CHR$ (136+j);
280 NEXT j
290 NEXT k
300 BRIGHT 0
310 FOR k=9 TO 12
320 FLASH 1
330 PRINT AT k,0;" ";
340 FOR j=0 TO 7
350 INK j
360 PRINT CHR$ (136+j);
370 PRINT CHR$ (136+j);
380 PRINT CHR$ (136+j);
390 NEXT j
400 NEXT k
410 FLASH 0
420 INK 0
430 PRINT AT 2,26;"Normal";
440 PRINT AT 6,26;"Bright";
450 PRINT AT 10,26;"Flash";
```

Fig 6.1 Demonstration of BRIGHT and FLASH.

steady colours. After FLASH 0 any symbols that are already flashing will continue to do so unless they are printed again.

The program listed in Fig 6.1 demonstrates the effects of using BRIGHT and FLASH.

Filling shapes with colour

One way of obtaining more boldly coloured pictures is to display patches or blocks of colour rather than single lines or dots. This can be done by filling in the area on the screen so that all of its dots are set at the INK colour.

Let us take a simple shape such as a rectangle. To fill the rectangle we can start by drawing the bottom side and then we can successively draw horizontal lines one above the other, equal to the width of the rectangle, until the top side of the rectangle is drawn.

```
100 REM Colour filled rectangle
110 FOR n=1 TO 20
120 CLS
130 INK INT (RND*7)
140 LET w=15+INT (RND*100)
150 LET x=INT (RND*(255-w))
160 LET h=10+INT (RND*60)
170 LET y=INT (RND*(175-h))
180 PLOT x,y
190 DRAW w,0
200 DRAW 0,h
210 DRAW -w,0
220 DRAW 0,-h
230 IF h>w THEN  GO TO 290
235 REM Fill with horizontal lines
240 FOR j=1 TO h
250 PLOT x,y+j
260 DRAW w,0
270 NEXT j
280 GO TO 330
285 REM Fill with vertical lines
290 FOR j=1 TO w
300 PLOT x+j,y
310 DRAW 0,h
320 NEXT j
330 PAUSE 50
340 NEXT n
```

Fig 6.2 Rectangle filling program

In the program listed in Fig. 6.2 the successive lines are drawn by first plotting the start point and then drawing a horizontal line equal to the width of the rectangle. For the next line the y co-ordinate is increased by 1 and the action is repeated. The process of filling is carried out in a loop which is executed h times, where h is the height of the rectangle.

There are two ways of filling a circle. In the first method a series of concentric circles is drawn with the radius of the circles being incremented one unit at a time until the desired radius is reached. This is shown in the program listed in Fig. 6.3.

```
100 REM Colour filled circles
110 REM using varying radius
120 FOR n=1 TO 25
130 LET r=INT (RND*50)
140 LET x=r+INT (RND*(255-2*r))
150 LET y=r+INT (RND*(175-2*r))
160 INK INT (RND*7): CLS
165 REM Draw circle outline
170 CIRCLE x,y,r
175 REM Fill circle
180 FOR j=0 TO r
190 CIRCLE x,y,j
200 NEXT j
210 PAUSE 50
220 NEXT n
```

Fig. 6.3. Filling a circle by varing radius.

```
100 REM Colour filled circles
110 REM using radial lines
120 FOR n=1 TO 25
130 LET r=INT (RND*50)
140 LET x=r+INT (RND*(255-2*r))
150 LET y=r+INT (RND*(175-2*r))
160 INK INT (RND*7): CLS
165 REM Draw circle outline
170 CIRCLE x,y,r
175 REM Fill circle
180 LET dt=PI/(r*4)
190 FOR j=0 TO r*8
200 PLOT x,y
210 DRAW r*COS (j*dt),r*SIN (j*dt)
220 NEXT j
230 PAUSE 50
240 NEXT n
```

Fig. 6.4. Filling a circle using radial lines.

The alternative approach to filling a circle is to draw radial lines from the centre and stepping the angle around the circle in small increments so that all of the dots in the circle are filled. This is shown in the program of Figure 6.4.

Filling a regular polygon, such as a hexagon, is best carried out by drawing a series of concentric polygons with the radius increasing one step at a time until the desired size is reached.

Different shades of colour

Using the INK and PAPER commands we can normally obtain eight different colours. The range can however be extended by using the BRIGHT command which in effect adds white to the selected colour thus producing a brighter symbol or dot on the screen and a paler shade of the basic colour. Thus BRIGHT applied to a red coloured symbol will produce a pink coloured symbol.

We can in fact produce a lot of new colours by mixing the basic

```
100 REM Colour mixing using
110 REM horizontal lines
120 FOR p=7 TO 0 STEP -1
130 FOR i=0 TO 7
140 PAPER 7
150 CLS
160 INK 0
170 PAPER p
180 PLOT 87,47
190 DRAW 82,0
200 DRAW 0,82
210 DRAW -82,0
220 DRAW 0,-82
230 INK i
240 FOR j=0 TO 77 STEP 2
250 PLOT 88,50+j
260 DRAW 79,0
270 PLOT 88,49+j
280 DRAW  PAPER p; INVERSE 1;79,0
290 NEXT j
300 PAUSE 50
310 NEXT i
320 NEXT p
```

Fig 6.5 Colour mixing using horizontal lines colours on the screen. If we draw a red line alongside a yellow line the result appears to be orange because at the normal viewing distance adjacent lines tend to merge together.

If we take a square and fill it with colour but draw alternate fill lines in a second colour we can achieve a simple form of colour mixing. The program listed in Fig. 6.5 shows the sort of results and the different shades of colour that can be produced.

```
100 REM Colour mixing using
110 REM alternate vertical lines
120 FOR p=7 TO 0 STEP -1
130 FOR i=0 TO 7
140 PAPER 7
150 CLS
160 INK 0
170 PAPER p
180 PLOT 87,47
190 DRAW 82,0
200 DRAW 0,82
210 DRAW -82,0
220 DRAW 0,-82
230 INK i
240 FOR j=0 TO 77 STEP 2
250 PLOT 90+j,48
260 DRAW 0,79
270 PLOT 89+j,48
280 DRAW  PAPER p; INVERSE 1;0,77
290 NEXT j
300 PAUSE 50
310 NEXT i
320 NEXT p
```

Fig. 6.6 Colour mixing using vertical stripes.

Colour mixing can equally well be carried out by using vertical stripes of alternate colour to fill the square. This is demonstrated by the program listed in Fig. 6.6. Here, however, you will probably find that with many combinations a series of patterning lines appear in the square and the colours produced may continually change giving a sort of flickering effect. A better result is achieved by using both horizontal and vertical lines as shown in the program listed in Fig. 6.7.

With alternate line colour mixing the results are fairly crude. A better technique is to produce an alternate dot pattern. The user defined graphics symbol facility can be used to create a special

```
100 REM Colour mixing using
110 REM crosshatch lines
120 FOR p=7 TO 0 STEP -1
130 FOR i=0 TO 7
140 PAPER 7
150 CLS
160 INK 0
170 PAPER p
180 PLOT 87,47
190 DRAW 82,0
200 DRAW 0,82
210 DRAW -82,0
220 DRAW 0,-82
230 INK i
235 REM Draw horizontal lines
240 FOR j=0 TO 79 STEP 2
250 PLOT 88,49+j
260 DRAW 78,0
270 NEXT j
275 REM Draw vertical lines
280 FOR j=0 TO 77 STEP 2
290 PLOT  PAPER p; INVERSE 1;89+j,49
300 DRAW  PAPER p; INVERSE 1;0,77
310 NEXT j
320 PAUSE 50
330 NEXT i
340 NEXT p
```

Fig. 6.7 Colour mixing by a crosshatch pattern

graphics symbol facility can be used to create a special character which contains a checkerboard pattern of dots as shown in Fig. 6.8.

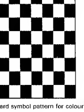


Fig. 6.8 Checkerboard symbol pattern for colour mixing.

Here the alternate dots both vertically and horizontally are in INK and PAPER colours so the two colours should be very effectively mixed within the symbol space. This technique is used in the program shown in Fig. 6.9 to produce a wide range of shades of colour.

```
100 REM Colour mixing by printing
110 REM crosshatch symbols
120 REM Create symbol
130 FOR n=0 TO 7 STEP 2
140 POKE USR "a"+n,170
150 POKE USR "a"+n+1,85
160 NEXT n
170 FOR p=7 TO 0 STEP -1
180 PAPER p
190 FOR i=0 TO 7
200 INK i
210 FOR b=0 TO 1
220 BRIGHT b
230 FOR r=5 TO 15
240 FOR c=10 TO 20
250 PRINT AT r,c;CHR$ 144;
260 NEXT c
270 NEXT r
280 PAUSE 50
290 NEXT b
300 BRIGHT 0
310 NEXT i
320 NEXT p
```

Fig. 6.9. Colour mixing using special symbols

Inverse Video

Normally symbols will be drawn in the INK colour on a background of the PAPER colour. By using INVERSE 1 symbols are displayed in the PAPER colour on a background INK colour. This can be useful for emphasising certain words in a message. It is possible to arrange that words are flashed on and off by alternately using INVERSE 1 and INVERSE ∅ display, but there is, in fact, a simpler method for getting flashing symbols by using the FLASH command.

Using the OVER command

An extremely useful colour command on the Spectrum is OVER. Like the FLASH and INVERSE commands it has two possible modes which are OVER 0 and OVER 1.

When OVER 1 is selected the new dot or symbol is added to anything that already exists on the screen at that particular character space.

The action of the OVER command is effectively an *exclusive OR* operation between the new data being written to the screen and that already being displayed. In an exclusive OR operation if a dot in the new pattern is set on or the equivalent dot in the screen picture is on then the dot displayed after the operation will be set on. Thus if the dot in the new symbol is in INK colour and the dot already on the screen is PAPER colour the result will be a dot set to INK colour.

If both dots are of the same colour, i.e. both PAPER or both INK, then the resultant dot on the screen is set at the PAPER colour. Thus we get the following four possible results.

Old dot	New dot	Result
PAPER	PAPER	PAPER
INK	PAPER	INK
PAPER	INK	INK
INK	INK	PAPER

The effect when two symbols are written to the same position using OVER is that where the two symbols overlap the dots are set to the PAPER colour, but all other dots in both symbols are displayed in INK colour. A point to note here is that if the earlier symbol had been written in a different colour its dots will now change to the current INK colour.

An interesting and very useful effect occurs if we write the same symbol on top of itself using the OVER function. The first time the symbol is written on to a blank space it will be displayed perfectly normally in the current INK colour. When the symbol is written again all of the dots in the new symbol coincide with those in the one already on screen so all of these dots are displayed in PAPER colour. Since all of the dots in PAPER colour also match up the whole character space will be in PAPER colour. In other words we have effectively erased the symbol.

Now this discovery may not seem particularly clever since we could have achieved exactly the same result by writing a blank character into the space. Suppose, however, that instead of starting with a blank we already had a character displayed there and that the new symbol that we are writing is a different one. On writing the new symbol for the first time with OVER we would get a combination of the two symbols in the space. Where the two symbols overlapped those dots would be set at the PAPER colour. Now if we write the second symbol again it will be erased wherever it does not overlap the earlier symbol. Where the symbols overlapped however the dots were at PAPER colour and these will now be set to INK colour thus restoring the original symbol on the screen. So we now have a way of writing and erasing a new character which has been written over an exiting one without erasing the original symbol.

Making a sketching program

So far in drawing our lines, triangles and rectangles we have had to calculate the positions of the points between which the lines are to be drawn. The corresponding x and y parameters are then used with the PLOT and DRAW commands to actually produce the line on the screen. In real life we would simply take a pencil or pen and just draw the line where we wanted it. It is not too difficult to achieve this on the Spectrum and we can produce a useful little sketching program.

In a simple sketching program the arrow keys can be used to move the graphics cursor position around the screen. Let us now assume that the graphics cursor position represents the position of the tip of a pen and that we can have the pen either on the paper (down) or lifted off the paper (up). When the pen is down it will draw a point on the screen and if it is moved when in the 'down' state a line will be drawn along the path of the pen tip. When the pen is up it does not draw on the screen but merely moves to a new position. The U (up) and D (down) keys are used to control whether the pen is up or down. At the start of the program the pen is set in the 'up' position and is located at the bottom left-hand corner of the screen where X=Ø and Y=Ø.

One problem which arises when we have the pen 'up' is that nothing is drawn so we have no means of knowing where the pen is. In the program this is overcome by placing a dot on the screen at the point where the pen is. Each time the pen moves the dot is erased and

```
100 REM Sketching program
110 PRINT "Arrow keys move pen up/down left/right"
120 PRINT "D key puts pen down for drawing"
130 PRINT "U key lifts pen for moving"
140 PRINT "E key erases line"
150 PRINT "Number keys 0 to 6 set colour"
160 PRINT "Keys C,V,8 and N give diagonal lines"
170 INPUT "Ready? (Y/N)";a$
180 IF a$<>"y" THEN  GO TO 170
185 REM Initialise
190 LET x=0: LET y=0: LET x1=0: LET y1=0
200 LET s=0: LET p=0: INK 0: PAPER 7
210 LET e=0: CLS : PLOT x,y
215 REM Test for key pressed
220 IF INKEY$="" THEN  GO TO 220
230 LET a$=INKEY$
235 REM Check for quit
240 IF a$="q" OR a$="Q" THEN  STOP
245 REM Check required pen state
250 IF a$="u" OR a$="U" THEN  LET p=0: GO TO 270
260 IF a$="d" OR a$="D" THEN  LET p=1: LET e=0
270 IF a$="e" OR a$="E" THEN  LET e=1
275 REM Check for arrow keys and set new x,y
280 IF a$=CHR$ 8 THEN  LET x=x-1: GO TO 360
290 IF a$=CHR$ 9 THEN  LET x=x+1: GO TO 360
300 IF a$=CHR$ 10 THEN  LET y=y-1: GO TO 360
310 IF a$=CHR$ 11 THEN  LET y=y+1: GO TO 360
315 REM Check for and set diagonal moves
320 IF a$="c" OR a$="C" THEN  LET x=x-1: LET y=y+1
330 IF a$="v" OR a$="V" THEN  LET x=x-1: LET y=y-1
340 IF a$="b" OR a$="8" THEN  LET x=x+1: LET y=y-1
350 IF a$="n" OR a$="N" THEN  LET x=x+1: LET y=y+1
355 REM Check x,y limits and execute wraparound
360 IF x>255 THEN  LET x=x-256
370 IF x<0 THEN  LET x=x+256
380 IF y>163 THEN  LET y=y-164
390 IF y<0 THEN  LET y=y+164
395 REM Set up colour
400 LET c=(CODE a$)-48
410 IF c>6 OR c<0 THEN  GO TO 430
420 INK c
425 REM Replace previous state if pen is up
430 IF p=1 OR s=1 THEN  GO TO 450
440 PLOT  OVER 1;x1,y1: OVER 0
450 LET s=POINT (x,y)
455 REM Carry out erasure
460 IF e=1 THEN  INVERSE 1
465 REM Plot new point
470 PLOT x,y
475 REM Update last point marker
480 LET x1=x: LET y1=y
485 REM Print current x,y position
490 PRINT AT 0,0;"x = ";x;" y = ";y;"    "
500 INVERSE 0
510 GO TO 220
```

Fig. 6.10 High resolution sketching program then redrawn in a new position so that it continually shows where the pen is at any time. If there is already a lit dot at the position before the pen moves to it this is noted by using the POINT command and the state of the pixel point is saved. When the pen move to a new position the original state of the pixel is restored. If this were not done then passing the pen over a line already drawn on the screen could cause part of that line to be erased.

Figure 6.10 gives a listing of this simple sketching program using keyboard control.

The arrow keys and the U and D keys are continuously monitored by using a loop and the INKEY$ command. When no key is pressed INKEY$ returns a blank string ("") and the test is repeated again. When a key is pressed a$ is set to INKEY$ and then is tested to see which key has been pressed and the appropriate action is taken. For the arrow keys to work correctly the CAPS SHIFT key must be held down whilst the arrow key is pressed. The program detects the actual code for the shifted arrow key. If desired, the approach used in the sketching program given in Chapter Two might be used. In that program the number keys corresponding to the four arrows were detected so that there was no need to use the CAPS SHIFT key. If an arrow key is held down the auto-repeat action of the Sinclair keyboard will come into play and a series of steps is drawn in succession to produce a line.

The program contains some further refinements. If the E key is pressed the pen will act as an eraser and will blank out any points that it passes over. This will allow the user to correct mistakes. Using the arrow keys will allow only horizontal or vertical lines to be drawn. Four extra keys are also recognised by the program. These are the C, V, B and N keys which are programmed to give diagonal movement to the pen. The N key moves up and to the right whilst the B key moves down to the right. The C and V keys move to the left and up and down respectively.

Colour attributes

In the Spectrum the text symbols are stored as dot patterns in the main video memory along with the dot patterns for the high resolution graphics pictures. Unlike other computers, however, the Spectrum stores its colour information in a separate area of memory. Thus the high resolution image is effectively stored as a pattern of black and white dots.

Colour information is not related to individual dots on the high resolution screen but to the character spaces, each of which consistds of an array of 8 x 8 or 64 dots The colours for all of the dots in this 8 × 8 array are determined by a single data word stored in a separate area of memory. This word provides what are known as the *attributes* for that symbol space on the screen.

The attribute word contains 8 data bits which are allocated as shown in Fig. 6.11. The lowest three bits of the attribute word give the INK colour for the symbol space. You will remember from Chapter One that colours are produced by combining the three primary colours, red, green and blue. The colour is set up by allocating the three bits to red, green and blue respectively. When only the red bit is on the symbol will be red but if red and green bits are both on then the displayed colour will be yellow (red + green) and so on.

Data Bit	Numerical Value	Colour Attribute
0	1	Blue INK
1	2	Red INK
2	4	Green INK
3	8	Blue PAPER
4	16	Red PAPER
5	32	Green PAPER
6	64	BRIGHT
7	128	FLASH

Fig 6.11 Bit assignment in the attribute word

The next three bits in the word are also allocated to red, green and blue but they indicate the PAPER colour information and is not related to individual dots on the high resolution screen but to the character spaces, each of which consists of an array of 8 + 8 or 64 dots. The colours for all of the dots in this 8 × 8 array are determined by a single data word stored in a separate area of memory. This word provides the attributes for that symbol space on the screen.

Reading the attributes

If we want to find out what the INK or PAPER colour is in a particular character space then we can do this by using the command:

LET a = ATTR(r,c)

where r and c are the row and column numbers for the chosen character space. The result is that a will now have the value of the attributes for that position on the screen.

In a 48K Spectrum the words indicating the attributes of the character spaces on the screen are stored in memory between locations 22528 and 23295. The attributes are stored in order starting with the attribute for location 0,0 at memory adress 22628. The following memory words are the attributes of the columns in row 0 working across the screen from left to right. The other rows then follow on in sequence. To locate a particular attribute we could use the calculation:

Memory address = 22528 + (32 × r) + c

If we wanted to know the colour of a particular space we could PEEK the corresponding location in the attribute area of memory and then decode the individual bits of the word to find out the INK and PAPER colours and whether the BRIGHT or FLASH conditions were selected. We could also set up a new attribute word or alter the existing one and POKE it back into the memory to change the colour in that space.

Often we may want to check the colour of an individual dot on the high resolution screen. This can be done by using the ATTR command in the form:

1ØØ a = ATTR (21–y/8,x/8)

Here, to find the column position, we have simply divided the x co-ordinate by 8 since there are 8 dots in each character space. The y calculation is a little more complicated because in the graphics mode y increases from bottom to top of the screen whereas in text mode rows are numbered starting from the top and working down. In this case we subtract y/8 from the available number of text rows to get the row number.

The attribute word may now be decoded by checking the state of each data bit in turn and setting a variable or "flag" to 1 or 0. Thus we start by checking to see if a is less than 128 in which case the FLASH bit must be 0 and we can set a variable FL = 0. If a>=128 then FL is set at 1, and 128 is subtracted from a ready for the next test. Variable a is now checked against 64 and a BRIGHT flag BR is set at 1 or 0 as appropriate. The other bits are then checked in sequence as shown in the listing of Fig. 6.12.

```
500 REM Decode colour attributes
510 LET a=ATTR (r,c)
515 REM Text FLASH bit
520 IF a<128 THEN LET fl=0: GO TO 540
530 LET fl=1: LET a=a-128
535 REM Test BRIGHT bit
540 IF a<64 THEN LET br=0: GO TO 560
550 LET br=1: LET a=a-64
555 REM Test PAPER bits
560 IF a<32 THEN LET pg=0: GO TO 580
570 LET pg=1: LET a=a-32
580 IF a<16 THEN LET pr=0: GO TO 600
590 LET pr=1: LET a=a-8
600 IF a<8 THEN LET pb=0: GO TO 620
610 LET pb=1: LET a=a-8
615 REM Test INK bits
620 IF a<4 THEN LET ig=0: GO TO 640
630 LET ig=1: LET a=a-4
640 IF a<2 THEN LET ir=0: GO TO 660
650 LET ir=1: LET a=a-2
660 LET ib=a
670 RETURN
```

Fig. 6.12 Subroutine to decode colour attributes

Chapter Seven
Graphs and Charts

By using a computer we can readily carry out lots of measurements or calculations and end up with enormous arrays of numbers. Having produced all of these numbers one method of presenting them is to produce a list or perhaps a table of figures. Unfortunately such a table or list of numbers is not particularly helpful when we come to interpret the results.

When examining a list of results we are usually more interested in the way the results are changing rather than the precise numbers. A much better method of displaying results is to show them visually using a graphics display or perhaps a chart. Such a graph or chart usually shows each result as either a varying length line or perhaps as a dot whose height above some reference line is proportional to the quantity being displayed. One of the simplest types of display is the variable length strip display and an example of this in real life is the everyday mercury thermometer.

Thermometer display

Let us start by looking at the production of a thermometer type display using the low resolution mosaic graphics symbols provided on the Spectrum.

In a conventional mercury thermometer the length of the column of mercury indicates the temperature. We can represent the mercury column by drawing a simple vertical bar whose length is proportional to the measurement it respresents, in this case temperature. The thermometer tube can be shown by drawing a box in a different colour around the measuring column. The height of this box must be sufficient to allow the measuring column to reach the maximum value that we want to display.

In order to make sense of the reading of a thermometer we need a scale. On a real thermometer this is normally drawn on, or alongside, the thermometer tube. On our display we shall draw the scale alongside the measurement column. Minus signs are used as graduation marks to show the calibration of the length of the column and some of these also have a number alongside which shows the corresponding temperature in degrees C. In this case only the lowest and highest temperature points are marked in this way. As the temperature changes the length of the vertical column changes in sympathy and the top of the column indicates the measured temperature.

Suppose we want to measure from 0°C to 100°C. The mosaic symbols allow us to draw in steps of half a text character space at a time so the maximum possible number of steps from the top to the bottom of the screen is only 44. A convenient length for the column might be 20 units. Each block in the column therefore represents 5°C. At this point we can draw the thermometer tube. The bottom of the tube is produced by printing mosaic symbols with codes 129,131 and 13ϕ roughly at the middle of text row 2ϕ. A loop is then used to draw the tube itself and the graduation marks by printing symbols in successive lines moving up from line 2ϕ. Finally the top of the tube is produced by printing three mosaic symbols on line 8 and the scale calibrations are printed at appropriate positions alongside the thermometer tube.

To draw the mercury column the temperature reading is first scaled into 5° steps by dividing t by 5. Note here that 5 is first added to t before it is scaled. This takes account of the fact that the − sign indicating 0°C is actually halfway up the lowest symbol position in the mercury column. After scaling the temperature value is rounded off and converted to an integer number y. Next a loop is set up with a limit of y/2 since there are two steps per symbol position. This loop prints completely filled character spaces working up from the bottom of the tube giving a length rounded down to the nearest 10°. Finally y/2 is compared with INT(y/2) to see if a further 5° step is needed and if so the next higher character space is filled with a half block symbol.

A program to produce the thermometer display on the low resolution screen is shown in Fig. 7.1. Random temperature readings are displayed as text at the top of the screen and also on the thermometer display. In this program before each new temperature is displayed the previous reading of the mercury column is erased by printing solid blocks in all of the column positions using INVERSE which effectively resets the column to the background or PAPER

```
100 REM Thermometer by mosaic graphics
110 CLS
120 INK 0: PAPER 7
130 LET xo=118: LET yo=16
140 REM Draw thermometer tube
150 PRINT AT 20,15;CHR$ 129;CHR$ 131;CHR$ 130;
160 FOR n=1 TO 11
170 PRINT AT 20-n,14;"-";CHR$ 133;CHR$ 128;CHR$ 138;
180 NEXT n
190 PRINT AT 8,15;CHR$ 132;CHR$ 140;CHR$ 136;
200 REM Draw Scale
210 PRINT AT 19,13;"0";
220 PRINT AT 9,11;"100";
230 PRINT AT 13,11;"C";
240 PRINT AT 14,10;"deg";
250 REM Display loop
260 FOR k=1 TO 100
270 LET t=INT (100*RND)
280 INK 1
290 PRINT AT 1,1;"Temperature = ";t;
300 PRINT " degrees C.    "
310 GO SUB 500
320 PAUSE 200
330 NEXT k
340 STOP
500 INVERSE 1
510 REM Erase previous reading
520 FOR n=1 TO 11
530 PRINT AT 20-n,16;CHR$ 143;
540 NEXT n
550 INVERSE 0
560 REM Draw new reading
570 INK 2
580 LET y=INT ((t+5)/5+0.5)
590 FOR n=1 TO INT (y/2)
600 PRINT AT 20-n,16;CHR$ 143
610 NEXT n
620 IF INT (y/2)=y/2 THEN  GO TO 650
630 LET y=INT (y/2)
640 PRINT AT 19-y,16;CHR$ 140;
650 RETURN
```

Fig. 7.1 Thermometer display using mosaic graphics

colour. The mercury column itself is drawn in red INK colour. The result on the screen is as shown in Fig. 7.2.

Of course the vertical column may be used to represent any quantity you like so this display could be used as a fuel gauge, speed indicator or even to indicate relative scores in a game. An alternative form of presentation would be to have the moving indicator strip horizontal so that it acts like the speedometer displays sometimes fitted to cars. In choosing the layout and screen position of these strip displays it is important to avoid having two different ink colours in any symbol space.

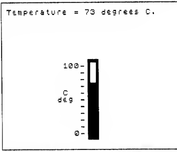


Fig 7.2 Typical display from the program in Fig 7.1

A better thermometer

A major problem with the thermometer display using the low resolution graphics mode is that it can only resolve quite large steps in the quantity being measured. By changing to the high resolution mode we can produce a rather more accurate readout. It is perhaps slightly easier to draw the tube and column using high resolution graphics but in order to add text to the display the graphics drawing needs to be carefully placed relative to the text symbol positions. This is also important to avoid colour problems since graphics colours are tied to symbol spaces.

The tube is easily drawn as a rectangle using PLOT and DRAW commands. Producing the scale marks is quite straightforward and uses DRAW commands in a loop. For convenience the scale mark for Ø is drawn separately before the start of the scale mark loop. The scale calibration values and the legend 'deg C' are simply printed at the appropriate positions by using PRINT AT commands.

Drawing the mercuy column involves producing a filled rectangle of height t units. The temperature scaling in this case is 1:1 and the maximum height of the mercury column is set at 100 screen units. With the high resolution thermometer there is no need for the 5° offset that we used for mosiac graphics since the scale graduation marks can be drawn at any required point on the screen. However the position of the tube does need to be chosen so that the text symbols line up with their calibration marks. The actual column is filled in by drawing six vertical lines alongside one another with each line of length t units. To take advantage of the DRAW command alternate lines are drawn up and down respectively relative to the cursor position and x is increased by one unit after each line is drawn.

A program to draw a thermometer style display using high resolution graphics is shown in Fig. 7.3 and the results on the screen are shown in Fig. 7.4. Of course the gauge can also be drawn with the moving measurement bar horizontal. This means rearranging the drawing sequence to produce horizontal lines instead of vertical

```
100 REM Hi-res thermometer
110 CLS
120 INK 0: PAPER 7
130 LET xo=118: LET yo=16
140 REM Draw thermometer tube
150 PLOT xo,yo
160 DRAW 10,0
170 DRAW 0,108
180 DRAW -10,0
190 DRAW 0,-108
200 REM Draw Scale
210 PLOT xo,yo+4
220 DRAW -3,0
230 DRAW 3,0
240 FOR n=1 TO 10
250 DRAW 0,10
260 DRAW -3,0
270 DRAW 3,0
280 NEXT n
290 PRINT AT 19,13;"0";
300 PRINT AT 6,11;"100";
310 PRINT AT 13,12;"C";
320 PRINT AT 14,11;"deg";
330 REM Display loop
340 FOR k=1 TO 100
350 LET t=INT (100*RND)
360 INK 1
370 PRINT AT 1,1;"Temperature = ";t;
380 PRINT " degrees C.     "
390 GO SUB 500
400 PAUSE 200
```

```
410 NEXT k
420 STOP
500 INVERSE 1
510 REM Erase previous reading
520 PLOT xo+2,yo+1
530 LET y=104
540 FOR n=1 TO 6
550 DRAW 0,y: DRAW 1,0
560 LET y=-y
570 NEXT n
580 DRAW 0,y
590 INVERSE 0
600 REM Draw new reading
610 INK 2
620 PLOT xo+2,yo+4
630 FOR n=1 TO 6
640 DRAW 0,t: DRAW 1,0
650 LET t=-t
660 NEXT n
670 DRAW 0,t
680 RETURN
```

Fig 7.3 High resolution thermometer display

ones and again the calibration numbers and text for labelling needs to be placed in appropriate positions relative to the actual measuring strip.

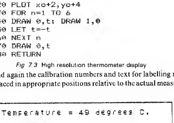


Fig 7.4 Typical thermometer display

In this program the temperature values are generated randomly by the computer and then displayed together with a printed readout of temperature at the top of the screen. By using a suitable input-output interface the Spectrum might be connected to an electronic thermometer. In this case the reading of temperature may be fed into the Spectrum and then displayed so that the screen display acts as if it were a real thermometer.

Bar charts

Whilst the thermometer style display is useful to show the current state of some measurement, a more useful arrangement would be to show how the situation varied over a period of time. We could perhaps measure the temperature at noon on each day of the week. A display showing this information can easily be arranged by drawing the thermometer displays for the days of the week alongside one another. For this display only the variable length bar is drawn for each day and a single scale is included at the left-hand side. To improve visibility the bars may be drawn with a gap between adjacent bars. This type of display is referred to as a histogram but is more commonly called a bar chart.

Bar charts are not normally intended to provide particularly accurate displays since their main application is to show the general trend of the variable being displayed. They are frequently used in business applications to show the trend in sales over a year, or perhaps the stock level, number of orders, or income over a period. It is very easy to see the trend of the results on such a chart.

A useful enhancement of the bar chart is to arrange that the colour of the bar is changed if its level goes above, or perhaps below, some predetermined limit. This can provide an easily recognised warning that a situation is becoming dangerous or needs attention. In such cases either the whole bar changes colour or the part above the limit line might change colour.

The low resolution mosaic graphics can be used to draw a bar chart since, although the vertical resolution is relatively coarse, the resultant display can be quite effective for this type of chart.

Figure 7.5 gives a listing for a program to draw a bar chart using mosaic graphics. In this program a separate bar is drawn for each day of the week and each bar is drawn using the same technique as for the mercury column in the thermometer program. The data in this program is read into an array so that the drawing of the bars can

```
100 REM Simple bar chart
110 REM using mosaic graphics
120 BORDER 3
130 INK 0: PAPER 7
140 DIM d$(7,2): DIM t(7)
150 REM Set up data
160 FOR n=1 TO 7
170  READ d$(n),t(n)
180 NEXT n
190 DATA "Mo",60,"Tu",65,"We",80
200 DATA "Th",55,"Fr",65
210 DATA "Sa",70,"Su",65
220 REM Draw scales
230 FOR n=1 TO 22
240 PRINT AT 19,7+n;CHR$ 131
250 NEXT n
260 FOR n=1 TO 11
270 PRINT AT 19-n,7;"-";CHR$ 138
280 NEXT n
290 FOR n=1 TO 7
300 PRINT AT 20,7+3*n;d$(n);" ";
310 NEXT n
320 PRINT AT 18,6;"0";
330 PRINT AT 8,4;"100";
340 PRINT AT 12,5;"F";
350 PRINT AT 13,4;"deg";
360 FOR j=1 TO 7
370 GO SUB 500
380 NEXT j
390 PRINT AT 2,10;"Daily Temperatures."
400 STOP
500 INK 2
510 REM Draw bar
520 LET y=INT ((t(j)+5)/5+0.5)
530 FOR n=1 TO INT (y/2)
540 PRINT AT 19-n,7+3*j;CHR$ 143;CHR$ 143;
550 NEXT n
560 IF INT (y/2)=y/2 THEN  GO TO 590
570 LET y=INT (y/2)
580 PRINT AT 19-y,7+3*j;CHR$ 140;CHR$ 140;
590 RETURN
```

Fig 7.5 Bar chart using mosaic graphics use a common drawing loop. It could easily be arranged that the temperature data is typed in from the keyboard by using an INPUT statement instead of READ to set up the temperature values.

The display produced on the screen is as shown in Fig. 7.6. By altering the scales and legends this program can readily be adapted to display any desired variable on the chart.

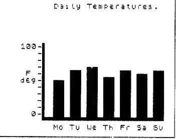


Fig. 7.6 Display produced by the program in Fig. 7.5

High resolution bar charts

Figure 7.7 shows a program to draw a bar chart using high resolution graphics and the result on screen is shown in Fig. 7.8. In this program the bars have been drawn in a different way from those of the thermometer. Here the loop limit is set to the desired reading in screen units and a series of short horizontal lines is drawn with one line above the other to produce the filled bar. This technique involves more passes around the loop than the vertical line version but is equally effective in producing bars. As in the case of the thermometer the position of the bars relative to the text symbol positions must be carefully chosen to avoid problems with display colour.

```
100 REM High res bar chart
110 CLS
120 BORDER 3
130 DIM d$(7,2): DIM t(7)
135 REM Set up data
140 FOR n=1 TO 7
150 READ d$(n),t(n)
160 NEXT n
170 DATA "Mo",15,"Tu",18,"We",25
180 DATA "Th",12,"Fr",17,"Sa",20,"Su",18
185 REM Draw axes and scales
190 INK 0
200 LET xo=48: LET yo=20
210 PLOT xo,yo
220 DRAW 168,0
230 PLOT xo,yo
240 FOR n=1 TO 6
250 DRAW 0,20
260 DRAW -3,0
270 DRAW 3,0
280 NEXT n
290 PRINT AT 20,7;;
300 FOR j=1 TO 7
310 PRINT d$(j);" ";
320 NEXT j
330 PRINT AT 19,2;"0";
340 PRINT AT 4,2;"30"
350 PRINT AT 11,2;"C";
360 PRINT AT 12,1;"deg"
365 REM Draw bars
370 INK 2
380 PLOT xo,yo
390 DRAW 4,0
400 FOR k=1 TO 7
410 DRAW 8,0
420 LET v=t(k)*4
430 FOR n=1 TO 4
440 DRAW 0,y
450 DRAW 1,0
460 DRAW 0,-y
470 DRAW 1,0
480 NEXT n
490 DRAW 8,0
```

```
500 NEXT k
510 REM Print legend
520 INK 1
530 PRINT AT 2,6;"Daily Temperatures";
540 STOP
```

Fig 7.7 High resolution bar chart program

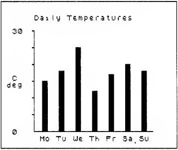


Fig 7.8 High resolution bar chart picture

Multiple bar charts

When two different variables are to be displayed on the same chart the bars are drawn in pairs so that they become interleaved. To provide clearer distinction between the sets of bars a different colour may be used for each set of bars. Three or perhaps four graphs could be interleaved in this way if desired. Some bars could be drawn as open boxes but with different coloured outlines. A typical application for a multiple bar chart might show the income and expenditure on a single chart. It might also be useful to show perhaps the predicted trends.

An example of a multiple bar chart is shown in the program listed in Fig. 7.9 which produces a plot of the maximum and minimum temperatures for the days of a week using high resolution graphics. In this case one set of bars is drawn in red whilst the others are in the

```
100 REM Multiple bar chart
110 CLS
120 BDRDER 3
130 D1M d$(7,2)
140 DIM 1(7)
150 D1M h(7)
160 REM Set up data
170 FOR n=1 TD 7
180 READ d$(n),1(n),h(n)
190 NEXT n
200 DATA "Mo",7,15,"Tu",10,18,"We"
210 DATA 15,25,"Th",5,12,"Fr",2,17
220 DATA "Sa",7,20,"Su",15,18
230 REM Draw axes and scales
240 INK 0
250 LET xo=48: LET yo=20
260 PLDT xo,yo
270 DRAW 168,0
280 PLDT xo,yo
290 FOR n=1 TO 6
300 DRAW 0,20
310 DRAW -3,0
320 DRAW 3,0
330 NEXT n
340 PRINT AT 20,7;;
350 FDR j=1 TD 7
360 PRINT d$(j);" ";
370 NEXT j
380 PRINT AT 19,4;"0";
390 PRINT AT 4,3;"30"
400 PRINT AT 11,2;"C";
410 PRINT AT 12,1;"deg"
420>REM Draw bars
430 PLOT xo+8,yo
440 FDR k=1 TO 7
450 INK 5
460 LET y=1(k)*4
470 FDR n=1 TO 4
480 DRAW 0,y
490 DRAW 1,0
500 DRAW 0,-y
```

```
510 DRAW 1,0
520 NEXT n
530 DRAW 4,0
540 INK 2
550 LET y=h(k)*4
560 FOR n=1 TO 4
570 DRAW 0,y
580 DRAW 1,0
590 DRAW 0,-y
600 DRAW 1,0
610 NEXT n
620 DRAW 4,0
630 NEXT k
640 REM Print legend
650 INK 1
660 PRINT AT 1,6;"Daily Temperatures";
670 INK 5
680 PRINT AT 3,6;"Low   ";CHR$ 143;
690 PRINT CHR$ 143;CHR$ 143;
700 INK 2
710 PRINT AT 3,17;"High   ";CHR$ 143;
720 PRINT CHR$ 143;CHR$ 143;
730 STOP
```

Fig 7.9 Multiple bar chart program

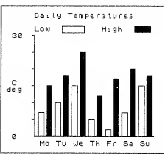


Fig 7.10 Typical multiple bar chart display

cyan colour. On such a chart a legend should always be included to show what each set of bars represents. Figure 7.10 shows the type of display produced by this program.

Bar graphs are often used in financial and production charts since they provide a bolder and easier to follow presentation than a list of figures.

Scientific graphs

Although the bar chart is well suited for business use, when we come to scientific or mathematical graph plotting a slightly different arrangement is used since the graph is required to give a more accurate display of results.

The layout is similar to that of a bar chart with the results of the calculation or experiment plotted vertically on the screen and the measurement steps horizontally. In this case however the value of Y is simply shown as a dot at a point equivalent to the top of the bar on a bar chart. Sometimes to make the point easier to see a small + sign, triangle or circle may be used as a marker instead.

In a bar chart the variables are normally positive but in a scientific graph the variables X and Y may be either positive or negative. To cater for this the X and Y axes are drawn as shown in Fig. 7.11.

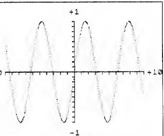


Fig. 7.11 A sine graph plot

Positive values of X are drawn to the right of the vertical Y axis and negative values of X to the left. Similarly positive values of Y are drawn above the X axis and negative values below. When there are no negative values for X the left-hand half of the graph is not drawn so that the Y axis appears at the left side of the diagram. Similarly if there are no negatve values of X only the upper part of the graph, above the X axis, would be drawn. Sometimes only a quarter of the complete graph need be drawn to display all of the points required. The advantage of drawing only part of the complete, X,Y axis system is that the required part of the chart can be expanded to fill the screen thus giving better resolution.

To see how this type of graph display is produced let us draw the graph for the equation Y = SIN(X) with values of X ranging from −1Ø to +1Ø. The value of SIN(X) will always lie between the limits −1 and +1 so this determines the scale of the Y axis. To produce a reasonable size graph we shall obviously have to multiply Y by a scaling value. A convenient scale is to arrange that 1 unit of Y gives 60 screen divisions, so we simply multiply the value of Y by a scaling factor ys before calculating the plotting co-ordinates. The value of ys is set at 60 at the start of the program. Similarly a multiplier xs is used to scale the X values. In this case xs is set at 10 at the start of the program to give a graph which is a total of 200 units wide on the screen. The values for xs and ys can be chosen to give the largest graph that will fit on the screen according to the range of values of x and y to be plotted.

The first step in constructing the graph is to produce the X and Y axis lines and scales. This can easily be done by using DRAW commands and two simple drawing loops. First we place the graphics cursor at the centre of the graph axes by using PLOT 128,88 which puts a dot at the centre of the screen. The next step is to draw the right-hand X axis which is built up by drawing a series of short horizontal lines each followed by a short vertical line going below the axis line and a second vertical line to take the cursor back on to the axis. The length of each step is the x increment multiplied by xs. After drawing the right-hand side the loop is repeated and the lines are drawn to the left. The Y axis and its scale marks are drawn in a similar fashion. The complete axis drawing stage is dealt with as a subroutine although it could equally well be done in line in the main program if desired. The last part of subroutine prints in the scale calibrations at each end of the axes.

Having drawn the axes, the next step is to plot the graph itself. Here the calculations are carried out in a loop with the angle (x) being stepped in small increments from −10 to +10. The x co-ordinate (xp) for each point is calculated from

xp = xo + (xs * x)

where xo is the x value for the centre of the graph which in this case is equal to 128.

Using a scaling factor xs allows the size of the graph plot on the screen to be easily altered and the value of xo may also be adjusted to place the graph in any desired position on the screen.

To plot the points on the graph the y value for the PLOT command is calculated from

yp = yo + ys * SIN(x)

and the point is then plotted using:

PLOT xp,yp

The program listing is given in Fig. 7.12 and the result produced on the screen is similar to Fig. 7.11. In this program the INK and PAPER colours are simply black and white but coloured graphs can be produced by changing the INK and PAPER colours to another combination.

```
100 REM Sine graph
110 BORDER 5
120 CLS
130 LET xs=10
140 LET ys=60
150 LET xo=128
160 LET yo=92
170 REM Draw axes
180 GO SUB 500
190 REM Plot graph
200 FOR x=-10 TO 10 STEP 0.1
210 LET y=SIN x
220 LET xp=xo+INT (xs*x)
230 LET yp=yo+INT (ys*y)
240 PLOT xp,yp
250 NEXT x
```

```
260 REM Print legend
270 PRINT AT 20,11;"Y = SIN(X)";
280 STOP
498 REM
499 REM Axis drawing subroutine
500 LET x=xs
510 REM Draw X axis
520 FOR k=1 TO 2
530 PLOT xo,yo
540 FOR j=1 TO 10
550 DRAW x,0
560 DRAW 0,-3
570 DRAW 0,3
580 NEXT j
590 LET x=-x
600 NEXT k
610 LET y=ys/10
620 REM Draw Y axis
630 FOR k=1 TO 2
640 PLOT xo,yo
650 FOR j=1 TO 10
660 DRAW 0,y
670 DRAW -3,0
680 DRAW 3,0
690 NEXT j
700 LET y=-y
710 NEXT k
720 PRINT AT 10,0;"-10";
730 PRINT AT 10,29;"+10";
740 PRINT AT 1,15;"+1";
750 PRINT AT 19,15;"-1";
760 RETURN
```

Fig 7.12 Program to draw sine graphics.

It would also be possible to draw two or more graphs on the same axes by using a different INK colour when plotting the dots for the second graph. One problem here is that where the dots of the first graph occupy the same character space as a dot from the second graph their colour will change to that of the second graph. In most cases this will present no real difficulty unless the lines of the two curves lie close to one another.

Joining the points

In order to obtain a good picture of the curve produced by the sine function a large number of values must be plotted so that the points are closely spaced. If there were less values for x and y the points would tend to be spread apart giving a less clear impression of the function shape.

Sometimes we may wish to find the probable value for y at a value of x that was not included in the points used for the graph. By using a technique known as interpolation we can obtain an approximate value for such an intermediate point on the curve.

The simplest technique for interpolation is to join successive points on the curve by straight lines. This is generally known as linear interpolation. We can in fact join the points with a straight line as the graph is plotted. This gives an easier to follow curve when the number of points available is limited. Some care is needed, however, because if too few points are used the straight line interpolation technique can be wildly inaccurate.

To join the points, the graph plotting routine is simply altered so that instead of using a PLOT command to plot each point a DRAW command is used to draw a short line from the last point plotted to the new point. A new pair of variables, x1 and y1, are now needed to specify the last point plotted. Variables x2 and y2 are used for each new point. After each line is drawn x1 and y1 are updated to equal the co-ordinates (x2,y2) of the latest point on the graph. The x,y values for the DRAW command are simply calculated by taking the difference between x2,y2 and x1,y1. The Spectrum will automatically move its graphics cursor to the new point as the line is drawn. The first point must be plotted using a PLOT command in order to place the graphics cursor in its required starting position on the graph. This is done by calculating an initial value for x1,y1 for the first point to be plotted. A variation of the simple graph plot program which uses interpolation to join the dots is shown in Fig. 7.13. In this program a cosine curve is plotted, which has the same shape as a sine curve but is shifted in position on the x axis as shown in Fig. 7.14.

```
100 REM Cosine graph with linked points
110 BORDER 2
120 CLS
130 LET xs=10
140 LET ys=60
```

```
150 LET xo=128
160 LET yo=92
170 REM Draw axes
180 GO SUB 500
190 REM Plot graph
200 LET x1=xo+xs*-10
210 LET y1=yo+INT (ys*COS -10)
220 PLOT x1,y1
230 FOR x=-10 TO 10 STEP 0.3
240 LET x2=xo+INT (xs*x)
250 LET y2=yo+INT (ys*COS x)
260 DRAW x2-x1,y2-y1
270 LET x1=x2
280 LET y1=y2
290 NEXT x
300 REM Print legend
310 PRINT AT 20,12;"Y = COS(X)";
320 STOP
498 REM
499 REM Axis drawing subroutine
500 LET x=xs
510 REM Draw X axis
520 FOR k=1 TO 2
530 PLOT xo,yo
540 FOR j=1 TO 10
550 DRAW x,0
560 DRAW 0,-3
570 DRAW 0,3
580 NEXT j
590 LET x=-x
600 NEXT k
610 LET y=ys/10
620 REM Draw Y axis
630 FOR k=1 TO 2
640 PLOT xo,yo
650 FOR j=1 TO 10
660 DRAW 0,y
670 DRAW -3,0
680 DRAW 3,0
690 NEXT j
700 LET y=-y
710 NEXT k
720 PRINT AT 10,0;"-10";
730 PRINT AT 10,29;"+10";
740 PRINT AT 1,15;"+1";
750 PRINT AT 19,15;"-1";
760 RETURN
```

Fig 7.13 Program to draw COS graph with linked points.

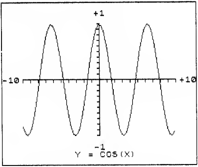


Fig 7 14 Display produced by COS program

Dial and clock type displays

For some applications a dial and pointer or clock type of display may be required. Typical uses might be in the instrument panel for a flight simulator program or perhaps to provide an instrument readout for an experiment where the computer is monitoring the results. In some cases, of course, the display may show time elapsed or time remaining in a games program.

The basic display consists of a circular, or possibly polygon shaped, dial with either one or two pointers. The pointers may simply be radially from the centre of the dial. The dial itself may also be filled with colour to make it stand out from the background. A scale of some sort is usually drawn around the outside of the dial.

This analogue type of display is often much more convenient where precise readings are not required but where the general trend can be taken in at a glance. An example of this is in digital and analogue watches and clocks. Although the digital display is precise it is much easier to tell the time by just glancing at a conventional clock face.

Drawing the dial is quite straightforward since it just involves drawing a circle which may be achieved by using the CIRCLE command. The scale is simply a larger circle. However to draw the scale gradations a further radius value is chosen. Thus s1 is the radius to the inner end of the scale marks whilst s2 is the radius to the outer ends of the marks. The position of each mark is calculated using rotation equations and then a short line is drawn radially out from the scale circle. A further radius is used to locate the position of the text symbols.

An important point about the hand or pointer is that it normally rotates clockwise for increasing readings. In fact the opposite will happen if we use the normal rotation equations. The position of the hand itself is readily calculated by using modified rotation equations. The angle of rotation required is simply the ratio of the scale reading to full scale multiplied by the total angle represented by full scale. If all 360 degrees are used, as in a compass display, then the angle TH is given by:

TH = 2*PI*X/FS

where X is the measured value, FS is the full scale reading and TH is the angle of rotation. To reverse the normal direction of motion the sign of the y term is reversed.

Sometimes the dial may cover only 90, 180 or 270 degrees. In this case the 2*PI term in the above equation should be reduced to the desired full scale angle measured in radians. The angle in radians is easily found by using the following equation:

RAD = DEG * PI / 18ϕ

Normally the rotation equation assumes that the zero point is horizontal and to the right. If you want zero to be at the top as in a compass (i.e. true north = 0) then 90 degrees or PI/2 must be added to values of TH before the values of x and y are calculated. Note that in this program this is achieved by using SIN in the x calculation and COS in the y calculation which produces the same effect as shifting through 90 degrees and changing the sign of y.

Drawing the scale marks is really similar to drawing the pointer except that the start of the mark line is at some radius a bit larger than the dial circle. The inner and outer ends of the mark are calculated using the rotation equation with two different values for r. The centre point of any text used for scale calibration can be calculated in the same way using a radius larger than the outer radius of the scale marks. Remember that having found the centre point for the text we have to write each symbol using DRAW and the appropriate shape string.

Usually the pointer will have to be redrawn for each new reading and the old pointer mark must be erased by redrawing it in the same colour as the dial fill or the background if the dial is not filled with colour. A procedure can be used to erase and redraw the pointer each time a new reading is calculated.

The program listed in Fig. 7.15 produces a simple dial display with a single pointer and gives a display similar to that shown in Fig. 7.16. In this program the dial starts with the pointer pointing up and has a 270 degree scale.

```
100 REM Moving pointer display
110 LET xc=128
120 LET yc=88
130 LET r=40
140 LET s1=r+5
150 LET s2=r+10
160 LET st=r+20
170 REM Draw dial
180 CIRCLE xc,yc,r
190 REM Draw scale
200 LET dt=1.5*PI/s1
210 LET th=0
220 LET x1=0
230 LET y1=s1
240 PLOT xc+x1,yc+y1
250 FOR n=1 TO s1
260 LET th=th+dt
270 LET x2=s1*SIN th
280 LET y2=s1*COS th
290 DRAW x2-x1,y2-y1
300 LET x1=x2
310 LET y1=y2
320 NEXT n
330 LET dt=1.5*PI/6
340 LET th=0
400 FOR n=0 TO 6
410 LET th=n*dt
420 LET x1=s1*SIN th
430 LET x2=s2*SIN th
440 LET y1=s1*COS th
```

```
 450 LET y2=s2*COS th
 460 PLOT xc+x1,yc+y1
 470 DRAW x2-x1,y2-y1
 480 LET xt=xc+st*SIN th
 490 LET yt=yc+st*COS th
 500 GO SUB 1000
 510 NEXT n
 520 REM Display pointer
 530 LET p1=0
 540 LET fs=6
 550 LET rp=r-5
 560 FOR k=1 TO 20
 570 FOR p=0 TO 6 STEP 0.2
 580 GO SUB 700
 590 NEXT p
 600 FOR p=6 TO 0 STEP -0.2
 610 GO SUB 700
 620 NEXT p
 630 NEXT k
 640 STOP
 690 REM Pointer subroutine
 700 LET th=1.5*PI*p1/fs
 710 REM Erase last reading
 720 INVERSE 1
 730 PLOT xc,yc
 740 DRAW rp*SIN th,rp*COS th
 750 INVERSE 0
 760 LET th=1.5*PI*p/fs
 770 REM Draw new pointer
 780 LET th=1.5*PI*p/fs
 790 PLOT xc,yc
 800 DRAW rp*SIN th,rp*COS th
 810 LET p1=p
 820 RETURN
 990 REM Text symbol subroutine
1000 PRINT AT 0,0;STR$ (n);
1010 REM Copy symbol
1020 FOR j=0 TO 7
1030 FOR i=0 TO 7
1040 IF POINT (i,175-j)=0 THEN  GO TO 1060
1050 PLOT xt+i-4,yt-j+4
1060 NEXT i
1070 NEXT j
1080 PRINT AT 0,0;" ";
1090 RETURN
```

Fig 7.15 Dial display program

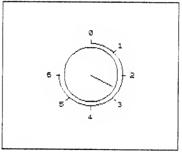


Fig. 7.16. Typical screen display for COS curve.

If two hands are required as in a conventional clock display then the same basic drawing routine may be used but with a different radius for each hand. Equally well a third pointer or hand might be added. If the pointers are to have different shapes it may be convenient to have a separate drawing procedure for each pointer.

Pie charts

A rather attractive form of display chart frequently used in business is the pie chart. This is used to show the proportions into which something divides up. An example might for instance be the percentage votes for political parties derived from a poll of a sample of electors. We have all seen these charts displayed on television. Another application might be to show how the resources of a company are used or how its money has been spent.

As its name implies the pie chart is effectively like a plan view of pie which has been sliced up into segments of various sizes. Each slice of the pie represents one item and shows the percentage of the total made up by that item. A typical pie chart is shown in Fig. 7.17.

To draw a pie chart we are effectively drawing a series of segments of a circle. The angle for each segment can be calculated as a percentage of 2 x PI (360 degrees). On the Spectrum the basic

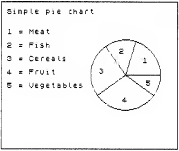


Fig. 7.17 Typical pie chart display.

technique is to start with drawing the outer circle by simply using the CIRCLE command. With the circle drawn the next step is to draw a series of radial lines which separate the slices of the pie. The X,Y offset values of the ends of these lines can be calculated using trigonometry as follows:

XR = DX*COS(TH)–DY*SIN(TH)
YR = DX*SIN(TH)+DY*COS(TH)
TH = P*2*PI/100

where P is the percentage of the complete pie represented by the segment being drawn. DX and DY are the offset co-ordinates for the last radial line drawn. A new set of rotated offset values XR and YR are calculated to allow the segment separation line to be drawn. After the segment has been drawn the values of DX and DY are updated to make them equal to XR and YR respectively ready for drawing the next sector.

A simple pie chart drawing program, allowing up to 5 segments, is shown in Fig. 7.18. The segment number or any other desired identification cannot readily be inserted by using a PRINT command because it is likely that the required position of the text symbol will not be one of the normal text symbol positions. To overcome this problem the identification number for the sector is placed in position by copying it dot by dot using the technique

```
100 REM Simple pie chart
110 DIM s(5)
120 REM Set up data
130 FOR n=1 TO 5
140 READ s(n)
150 NEXT n
160 DATA 20,15,25,30,10
170 REM Draw circle
180 LET xc=180
190 LET yc=88
200 LET r=50
210 CIRCLE xc,yc,r
220 REM Mark off sectors
230 LET dx=r
240 LET dy=0
250 LET th=0
260 FOR n=1 TO 5
270 LET xr=dx*COS th-dy*SIN th
280 LET yr=dx*SIN th+dy*COS th
290 REM Draw sector line
300 PLOT xc,yc
310 DRAW xr,yr
320 LET dx=xr
330 LET dy=yr
340 REM Write in sector number
350 LET th=PI*s(n)/100
360 LET xr=dx*COS th-dy*SIN th
370 LET yr=dx*SIN th+dy*COS th
380 LET xt=xc+INT (.7*xr)
390 LET yt=yc+INT (.7*yr)
400 LET a$=STR$ (n)
410 GO SUB 600
420 LET dx=xr
430 LET dy=yr
440 NEXT n
450 REM Print legend
460 PRINT AT 1,1;"Simple pie chart";
470 PRINT AT 4,1;"1 = Meat";
480 PRINT AT 6,1;"2 = Fish";
490 PRINT AT 8,1;"3 = Cereals";
500 PRINT AT 10,1;"4 = Fruit";
```

```
510 PRINT AT 12,1;"5 = Vegetables";
520 STOP
600 PRINT AT 0,0;a$;
610 FOR j=0 TO 7
620 FOR i=0 TO 7
630 IF POINT (i,175-j)=0 THEN GO TO 650
640 PLOT xt+i-4,yt-j+4
650 NEXT i
660 NEXT j
670 PRINT AT 0,0;" ";
680 RETURN
```

Fig 7.18 Program to draw pie chart

described in Chapter Five. The symbol required is actually printed at position 0,0 to provide a dot pattern for copying. The co-ordinates for the character are calculated for its top left corner by adding a small offset to the co-ordinates of the point at the centre of the segment where the symbol is to be placed. The position of this number is calculated by making the rotation for the sector in two parts. First a rotation of half the sector angle is made then the text symbol is drawn and then the remainder of the rotation is carried out. This places the test symbol roughly in the middle of the sector.

When numbers or letters are used to identify the sectors some sort of key showing what the sectors represent should be included on teh chart. This can of course be printed normally by using PRINT or perhaps PRINT AT.

Chapter Eight
The World in Motion

For many computer games and particularly those of the arcade type we shall want to produce moving objects on the display screen. As we saw in Chapter One the television display presents twenty-five complete pictures in rapid succession every second and because of the lag in response of our eyes we do not see the flicker as the pictures are traced out. If we arrange that an object moves to a slightly different position on each successive scan of the television screen, then to the viewer it will appear to move around the screen. This is the basic principle used in producing cartoon films. To achieve reasonably smooth movement we should make the changes between pictures at least once every tenth of a second. If the movement steps are small the movement will appear to be smooth but with larger steps between successive pictures the motion will tend to become jerky.

In the simplest form of animation an object such as a ball, alien invader or spaceship is moved from one position to another on the screen. For more realistic results the object on the screen may need to change shape as it moves. An example of this would be a man walking across the screen. If the image of the man remained constant he would appear to glide across the screen rather like an ice skater. To give the impression of walking or running the position of the legs, and perhaps the arms too, of the man must be changed as the image moves from one position to another on the screen. In effect we will present a series of slightly different images in rapid succession. Most actions, such as walking are repetitive so we could perhaps have three or four different images and just repeat the sequence as the man moves across the screen. For a bird flying on the screen the wings will need to flap but in real life this is not just the simple up and down motion that we might imagine. In fact the whole shape of the bird's wing changes as it makes a beat in the air and for realistic results we would have to produce similar shape changes.

A simple moving ball

For many games-type programs a simple object such as a ball moves around the screen. One possibility here is to use a text symbol such as an asterisk for the ball and then print it in a new character position alongside its present position. In order to avoid leaving a trail of symbols across the screen we then need to erase the symbol at the previous position. This is easily done by printing a space symbol over it.

Let us start off by looking at the process involved in displaying a simple moving ball which we shall represent by using the solid graphics block symbol. This symbol has the character code 143. The position of the ball can be set up by using two variables x and y to give the screen position co-ordinates for the ball. We might start by placing the ball at the centre of the screen with x=15 and y=10 by using

```
190 PRINT AT y,x,CHR$ 143,
```

Making the ball move horizontally across the screen is quite straightforward since all we have to do is add one to the x value to move the ball to the right or subtract one from x to move the ball left. Thus we get a new x value, which we shall call xn, and this is now used in a PRINT AT statement to print the ball symbol in its new position. You may wonder why we didn't alter the value of x. One reason for this is that we still need x to allow us to print a blank space over the previous symbol in order to erase it. Before the next move is made the value of x is set equal to xn and then we are ready to carry out the whole process again for the next ball movement.

For vertical motion we need to change the y co-ordinate. Adding one to y will move the ball down the screen and subtracting one from y moves it up the screen. Once again we can use a second variable yn for the new y position. As for the horizontal motion we can print the ball in its new position and then erase the symbol at the previous position by overprinting it with a space. After the erasure step the y variable is set equal to yn ready for the next step.

Diagonal motion is a slightly more complicated process since we have to alter both x and y values at each step. If we want to move the dot up and to the right the new values for xn and yn will be x+1 and y-1 respectively. Up to the left is x−1, y−1 and for the downward motions we will need y+1 with either x−1, for moving left or x+1 for moving right. Figure 8.1 shows the values required for xn and yn for all eight directions assuming that the ball is currently positioned at

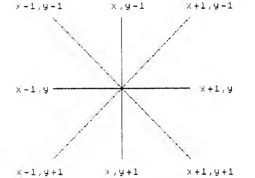


Fig 8.1 Change in X,Y for movement in various directions.

x, y. In these horizontal, vertical or diagonal moves the values of x and/or y are altered one at a time.

For higher speed movement we could change the values of x and y by more than one unit at a time. To allow for this possibility it is convenient to use two more variables dx and dy to represent the difference in x and y position for each step. There is another advantage in using the dx and dy terms. It becomes very easy to reverse the direction of motion by simply setting dx=−dx or dy=−dy.

Bouncing off the walls

If you tried moving the ball using the simple program sequence we have just discussed problems would soon arise. Suppose we started with the ball at the centre of the screen and then moved it step by step to the right. All will be well until the ball passes the edge of the screen and the value of x becomes 32. At this point the program will stop and an error message will appear. The reason is that x value has gone beyond its permissible limit value which is 31. A similar situation will occur if we were moving in the opposite direction and x became negative. When the ball is moving vertically errors will occur if y goes negative or is greater than 21. To avoid this state of affairs we must arrange that when the ball reaches one of the edges this is detected and the direction of motion is reversed. The effect of this is that when the ball reaches an edge of the screen it is reflected back towards the middle as if it has bounced off a wall.

This bouncing action is quite easy to achieve. Having calculated the values xn and yn for the next position of the ball these values are compared with the limit values for x and y. Let us start with the x value. Here a simple IF statement checks the value of xn to see if it is equal to ∅ or 31. If the test result is true the sign of the dx is changed by setting dx=−dy. This prevents the ball from going off the screen since, when a new value of xn is calculated by adding the new version of dx to xn, to set up the next move the ball position will move in the opposite x direction going towards the middle of the screen. This basic technique will work even if dx is greater than 1 provided that the IF test checks for xn<=∅ or xn>=31 and the new dx is added to xn before printing the symbol.

The same basic process may be applied to the yn value and in this

```
100 REM Simple moving ball
110 BORDER 1
120 PAPER 4
130 CLS
140 LET x=15
150 LET y=10
160 LET dx=1
170 LET dy=1
180 INK 7
190 PRINT AT y,x;CHR$ 143;
200 REM Movement loop
210 LET xn=x+dx
220 LET yn=y+dy
230 IF xn=0 OR xn=31 THEN  LET dx=-dx
240 IF yn=0 OR yn=21 THEN  LET dy=-dy
250 PRINT AT y,x;" ";
260 PRINT AT yn,xn;CHR$ 143;
270 LET x=xn: LET y=yn
280 GO TO 210
```

Fig 8.2 Simple moving ball symbol

case dy is changed in sign if yn<=∅ or yn<=21. Once again if dy changes sign it is added to yn before printing the symbol. After these checks have made the symbol at the old position is erased and ball is printed at its new position. Finally x and y are updated ready for the next step. This action is shown in the program listed in Fig. 8.2. When run this will show a block which starts off at the screen centre and then travels diagonally bouncing off the sides of the screen when it reaches them.

Reflection off a bat

In games such as SQUASH or BREAKOUT the ball bounces off a movable bat and depending upon where it hits the bat will travel straight or diagonally alter reflection. Here we need to keep track of the bat position by using two new variables bx and by.

Now in addition to the tests for reflection off a wall routine we need to include a check to see if the new position of the ball (xn, yn) is equal to that of the bat (bx,by). If the two positions are equal then the reversal of direction is arranged by reversing dy if the bat is at the top or bottom of the screen or dx if the bat is at the side of the screen. At the same time a score might be updated.

Bats are usually made wider than one character space and a typical scheme might be to have the bat three symbols wide. In this case we also need to check the ball position against bx+1,by and bx+2, by. This assumes that the left symbol of the bat is at position bx. Often the motion of the ball after it hits the bat is determined by where it hits the bat. So if the centre of the bat is hit (i.e. xn, yn=bx+1) by then dx would be set at ∅ and dy at −1 so that the ball travels straight up the screen. This assumes that the bat is in the lower side of the screen. If one of the other segments of the bat is hit then diagonal motions are produced by setting dx to +1 or −1 and dy to −1. In most games of this type, such as BREAKOUT, the bottom wall, containing the bat, may be checked for coincidences with the ball and if there is a match the ball is lost and the game ends or a new ball is used up. In this case the check is simply made for coincidence between yn and 21 and if this is true a further check is made against bx to see if the ball has hit the bat.

Moving the bat

The most convenient method of determining bat position on the Spectrum is to use two adjacent keys on the keyboard one giving

```
100 REM Simple squash game
110 BORDER I
120 LET hs=0
130 LET bx=15: LET by=20
140 LET bx1=bx: LET byI=by
145 REM Set up new game
150 CLS
160 LET nb=0
170 LET x=15: LET y=9
180 LET sc=0
190 LET b$=CHR$ 143+CHR$ 143+CHR$ 143
200 PRINT AT 21,0;"Press space to serve      ";
210 LET a$=INKEY$: IF a$<>" " THEN GO TO 2I0
215 REM Serve new ball
220 LET x=I5: LET y=9
230 PRINT AT y,x;CHR$ 143;
240 LET dx=INT (RND*3)-1: LET dy=-1
245 REM Main play loop
250 PRINT AT 21,0;"Score= ";sc;
260 PRINT "       Hi score= ";hs;"      ";
265 REM Move bat
270 LET m$=INKEY$
280 IF m$="z" THEN  LET bx1=bx-1
290 IF m$="x" THEN  LET bx1=bx+1
300 IF bx1<0 THEN  LET bx1=0
310 IF bx1+2>31 THEN  LET bxI=29
320 PRINT AT by,bx;"   ";
330 PRINT AT byI,bx1;b$;
340 LET bx=bx1: LET by=by1
345 REM Move ball
350 LET xn=x+dx: LET yn=y+dy
355 REM Test for walls
360 IF xn=0 OR xn=31 THEN  LET dx=-dx
370 IF yn=0 THEN  LET dy=-dy
375 REM Test for bottom of screen
380 IF yn=by THEN  GO TO 450
390 PRINT AT y,x;" ";
400 PRINT AT yn,xn;CHR$ I43;
405 REM Update ball position
410 LET x=xn: LET y=yn
420 GO TO 250
440 REM Test for hit by bat
450 IF xn=bx THEN  GO TO 550
460 IF xn=bx+I THEN  GO TO 550
470 IF xn=bx+2 THEN  GO TO 550
480 LET nb=nb+1
485 REM Test for game end
490 IF nb=5 THEN  GO TO 1000

500 PRINT AT y,x;" ";
510 GO TO 200
540 REM Set new direction
550 LET dx=-dx: LET dy=-1
560 IF xn=bx AND bx<>0 THEN  LET dx=-I
570 IF xn=bx+2 AND bx<>29 THEN  LET dx=I
575 REM Update score
580 LET sc=sc+I
```

```
 590 GO TO 250
 990 REM End of game
1000 BEEP 1,12
1010 PRINT AT 0,0;"GAME OVER"
1015 REM Check for high score
1020 IF hs>sc THEN  GO TO 1050
1030 PRINT  FLASH 1;AT 2,0;"*NEW HIGH SCORE*"
1040 LET hs=sc
1050 INPUT "Another game";g$
1060 IF g$="y" THEN  GO TO 130
1070 STOP
```

Fig 8.3 Simple squash style game

movement to the right and the other to the left. We could in fact use the arrow keys for this but for the moment let us use the Z key to move left and the X key to move right. The technique for using these keys is similar to that used for the sketching programs in Chapters Two and Six.

At this point we can devise a simple game where the score is increased by one each time the ball is hit by the bat, and the game ends after five balls have been played. Each new ball starts off from the centre of the screen travelling upwards. The program listing is shown in Fig. 8.3. Note here that the upper limit of the screen has been set at y=2 to allow the score to be printed on the top line.

Animation using high resolution

One disadvantage of using the text or low resolution display is that because the steps of movement of the object are quite large the resultant motion tends to look rather jerky. If we move to the high resolution screen the situation becomes better.

On the high resolution screens the basic principles for moving an object are still the same. There is an important difference between printing a symbol and plotting a high resolution dot. The Y value on a text screen starts at 0 at top of the screen and increases as we move down the screen whereas a dot will move up the screen for increasing values of y. Changing the values of x and y for the object will now produce different directions of motion and these are shown in Fig. 8.4. On the high resolution screen a single step in any direction moves the object a shorter distance and the resultant motion looks smoother. The program listed in Fig. 8.5 demonstrates a simple moving dot on the high resolution screen.

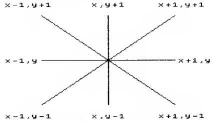


Fig. 8.4. Change in x,y required for high resolution

```
100 REM Moving hi-res dot
110 BORDER 6
120 LET x=128
130 LET y=88
140 LET dx=2
150 LET dy=2
170 PLOT x,y
180 LET xn=x+dx
190 LET yn=y+dy
195 REM Check screen limits
200 IF xn>0 AND xn<255 THEN  GO TO 220
210 LET dx=-dx: LET xn=xn+dx
220 IF yn>0 AND yn<175 THEN  GO TO 240
230 LET dy=-dy: LET yn=yn+dy
235 REM Erase last dot
240 PLOT  OVER 1;x,y
245 REM Plot new dot
250 PLOT  OVER 1;xn,yn
260 LET x=xn
270 LET y=yn
280 GO TO 180
```

Fig. 8.5. Simple moving high resolution dot

Moving more complex objects

So far we have just tried moving a dot around the screen but of course we shall usually want to move something rather more complex such as perhaps a flying saucer. This is demonstrated in the

```
100 REM High res flying saucer
110 PLOT 0,0
120 DRAW 255,0
130 DRAW 0,175
140 DRAW -255,0
150 DRAW 0,-175
160 LET x=128: LET y=88
170 LET x1=x: LET y1=y
180 LET dx=2: LET dy=dx
190 OVER 1
200 GO SUB 500
210 FOR n=1 TO 500
220 LET x1=x+dx: LET y1=y+dy
230 IF x1+dx<6 OR x1+dx>248 THEN  LET dx=-dx
240 IF y1+dy<6 OR y1+dy>168 THEN  LET dy=-dy
260 GO SUB 500
270 LET x=x1: LET y=y1
280 GO SUB 500: NEXT n
500 PLOT x,y-2: DRAW 3,0: DRAW 2,2
510 DRAW -2,2: DRAW -1,0: DRAW -2,2
520 DRAW -2,-2: DRAW -1,0: DRAW -2,-2
530 DRAW 2,-2: DRAW 3,0: RETURN
```

Fig. 8.6. Flying saucer program

program listed in Fig. 8.6.

The flying saucer itself is produced by a single PLOT command and a series of DRAW commands in a subroutine. The main program calculates an x,y position for the saucer and then calls the subroutine to draw the saucer. The OVER1 command is set up before drawing starts. To erase the saucer it simply has to be drawn again in the same position. In this program the saucer is always drawn at position x,y. When the new position x1, y1 has been calculated the saucer is redrawn to erase the image at x,y and then x and y are updated to x1, y1 and the saucer drawn again.

As in the case of the moving ball and moving dot the movement step is set up using the two terms dx and dy. When a new position has been calculated it is compared with the screen limits, then dx and dy are altered if required to make the saucer bounce off the screen boundary. Note in this case the screen limits are set in from the edge of the screen to allow for the width and height of the saucer. Remember that the saucer position is measured at the middle of the saucer figure.

One problem you will notice is that the drawing process is relatively slow in BASIC and this makes this type of animation rather limited since it must inevitably be slow unless you start writing the program in machine code. This is why most of the fast action games for the Spectrum are written in machine code or at least the movement routines are in machine code.

Animation using special graphics symbols

There is another approach to animation which uses normal PRINT techniques but gives smoother motion. This involves the use of special custom designed graphics symbols.

Suppose we want to move a diamond shaped object across the screen. We could start by creating a symbol for the required shape as decribed in Chapter Five. Now suppose we want to move the shape two dot positions at a time across the screen. At the second position part of the object will have moved into the following character space. We can handle this quite easily by simply creating two new symbols which when printed one after the other will show the diamond in its new position. For the next step a further pair of symbols is created where the diamond is halfway between the two symbol positions. The fourth position has the diamond mostly in the following character space. At the fifth step the symbol will actually be in the next character space and we can start the whole process again but this time one character position further across the screen.

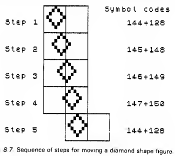


Fig. 8.7 Sequence of steps for moving a diamond shape figure.

This sequence of events is shown in Fig. 8.7 and a program for moving a diamond shape across the screen is shown in Fig. 8.8.

This technique can readily be applied to more complex objects. Suppose we had a flying saucer which took up two adjacent spaces on the screen. We should now need a pattern group of three successive symbols. Here again when the object has moved out of the first symbol position of the group then the whole pattern is moved by one symbol space and the animation action is repeated.

```
100 REM Moving diamond shape
110 REM Set up symbols
120 GO SUB 500
125 REM Movement loop
130 FOR m=1 TO 20
140 FOR c=1 TO 20
150 PRINT AT 10,c;CHR$ 144;CHR$ 128;
160 PRINT AT 10,c;CHR$ 145;CHR$ 148;
170 PRINT AT 10,c;CHR$ 146;CHR$ 149;
180 PRINT AT 10,c;CHR$ 147;CHR$ 150;
190 PRINT AT 10,c;CHR$ 128;CHR$ 144;
200 NEXT c
210 NEXT m
220 STOP
490 REM Set up characters
500 POKE USR "a",BIN 00001000
510 POKE USR "a"+1,BIN 00010100
520 POKE USR "a"+2,BIN 00100010
530 POKE USR "a"+3,BIN 01000001
540 POKE USR "a"+4,BIN 00100010
550 POKE USR "a"+5,BIN 00010100
560 POKE USR "a"+6,BIN 00001000
570 POKE USR "a"+7,0
580 POKE USR "b",BIN 00000010
590 POKE USR "b"+1,BIN 00000101
600 POKE USR "b"+2,BIN 00001000
610 POKE USR "b"+3,BIN 00010000
620 POKE USR "b"+4,BIN 00001000
630 POKE USR "b"+5,BIN 00000101
640 POKE USR "b"+6,BIN 00000010
650 POKE USR "b"+7,0
660 POKE USR "c",BIN 00000000
670 POKE USR "c"+1,BIN 00000001
680 POKE USR "c"+2,BIN 00000010
690 POKE USR "c"+3,BIN 00000100
700 POKE USR "c"+4,BIN 00000010
```

```
710 POKE USR "c"+5,BIN 00000001
720 POKE USR "c"+6,BIN 00000000
730 POKE USR "c"+7,0
740 POKE USR "d",0
750 POKE USR "d"+1,0
760 POKE USR "d"+2,0
770 POKE USR "d"+3,1
780 POKE USR "d"+4,0
790 POKE USR "d"+5,0
800 POKE USR "d"+6,0
810 POKE USR "e",0
820 POKE USR "e"+1,0
830 POKE USR "e"+2,BIN 10000000
840 POKE USR "e"+3,BIN 01000000
850 POKE USR "e"+4,BIN 10000000
860 POKE USR "e"+5,0
870 POKE USR "e"+6,0
880 POKE USR "e"+7,0
890 POKE USR "f",BIN 10000000
900 POKE USR "f"+1,BIN 01000000
910 POKE USR "f"+2,BIN 00100000
920 POKE USR "f"+3,BIN 00010000
930 POKE USR "f"+4,BIN 00100000
940 POKE USR "f"+5,BIN 01000000
950 POKE USR "f"+6,BIN 10000000
960 POKE USR "f"+7,0
970 POKE USR "g",BIN 00100000
980 POKE USR "g"+1,BIN 01010000
990 POKE USR "g"+2,BIN 10001000
1000 POKE USR "g"+3,BIN 00000100
1010 POKE USR "g"+4,BIN 10001000
1020 POKE USR "g"+5,BIN 01010000
1030 POKE USR "g"+6,BIN 00100000
1040 POKE USR "g"+7,0
1050 RETURN
```

Fig. 8.8 Program to draw moving diamond.

Collisions with other objects

In many games types programs, such as space invaders the object is to fire missiles or drop bombs on other objects to destroy them. This means that we must detect when the missile reaches the same position as a target object. One technique for this is to maintain a table of the x,y positions of the objects on the screen. Before moving the missile to its new position a check is made by comparing this position with that of each of the other objects in turn. If a match occurs the program will branch to a subroutine or procedure which produces the required explosion effect and erases both the missile and the target object. If there are a lot of objects on the screen this can become quite a complex process.

A simpler technique that can be used for detecting when a missile hits an object is to use POINT to check if the next position to which the missile is to be moved is already in INK colour. If not, then the missile move is made as normal. If the POINT command detects a dot in INK colour then the missile is about to hit an object and the program can be made to branch off to a hit routine. This might blank out the target object and missile and then replace the target object with an explosion effect, suitably accompanied by sound of course. Finally the explosion image is blanked and a new game sequence starts.

Animation involving shape changes

So far in our experiments with animation the object being moved stays the same shape as it moves across the screen. This is, of course, perfectly all right for things such as balls, flying saucers and the like. When we come to displaying things such as aliens, or men, however, the situation is a little different.

If we drew a figure of a matchstick man and just moved it across the screen in the same way as we move the saucer it would look as if the man were gliding across the screen because his legs and arms do not change position as he moves. Even the aliens of a typical invaders type game have their legs moving as they march across the screen.

The technique for producing changes in the shape of an object rely on the use of two or more different versions of the object being animated. Let us start by taking a relatively simple alien invader. We want the legs to move and a fairly simple motion would have the legs pointing inward on one step and outward on the next step. The first stage therefore is to create special symbols for two separate pictures of our alien one with the legs together and the other with them apart.

Now to produce the required animation we draw the first of the two shapes. For the next step we calculate the next position of the alien then erase the first shape and draw the second shape at the new position. For the next step we draw the first shape again and so on, alternating the shapes as the alien moves across the screen. This is shown in the program listed in Fig 8.9.

Now for a walking man we have to move to a new level of complexity. In this case four separate shapes are created and set up as a strings. Once again as the man moves across the screen the four pictures of the man are drawn and erased in sequence to produce the effect of a walking man. Better results could be obtained by using more intermediate pictures to form each step that the man makes. Here we have to make a trade off between using a lot of different images to give smooth action and the speed of movement and amount of memory used. The more steps there are the longer it takes for the man to make one step forward. A compromise can usually be reached where the action is reasonably realistic but not too complex or too slow. Remember that a fast moving object does not need to have its action so accurately portrayed because its speed covers up many inadequacies in the shape changes used.

Animation of objects where the shape changes, and particularly if they are familiar natural objects, usually involves some study of the way that things move in real life and then a simplified version is used to animate the computer drawn object. In fact the process of animation is an art form in itself and much fun can be had by experimenting with different ideas. Here we have outlined the principles involved and showed some of the techniques used in animating objects on the computer graphics screen.

```
100 REM Moving alien
110 REM with shape change
120 REM Set up symbols
130 GO SUB 500
135 REM Animation sequence
140 FOR p=20 TO 0 STEP -2
150 LET r=10
160 FOR c=0 TO 28 STEP 2
170 PRINT AT r,c;CHR$ 144;CHR$ 145;
180 PAUSE p
190 PRINT AT r,c;"  ";
200 PRINT AT r,c+1;CHR$ 146;CHR$ 147;
210 PAUSE p
220 PRINT AT r,c+1;"  ";
230 NEXT c
240 NEXT p
250 STOP
```

```
490 REM Set symbols subroutine
500 POKE USR "a",BIN 00111110
510 POKE USR "a"+1,BIN 01000001
520 POKE USR "a"+2,BIN 01011100
530 POKE USR "a"+3,BIN 01000000
540 POKE USR "a"+4,BIN 00111111
550 POKE USR "a"+5,BIN 00100100
560 POKE USR "a"+6,BIN 01001000
570 POKE USR "a"+7,BIN 10010000
580 POKE USR "b",BIN 01111100
590 POKE USR "b"+1,BIN 10000010
600 POKE USR "b"+2,BIN 00111010
610 POKE USR "b"+3,BIN 00000010
620 POKE USR "b"+4,BIN 11111100
630 POKE USR "b"+5,BIN 00100100
640 POKE USR "b"+6,BIN 01001000
650 POKE USR "b"+7,BIN 10010000
660 POKE USR "c",BIN 00111110
670 POKE USR "c"+1,BIN 01000001
680 POKE USR "c"+2,BIN 01011100
690 POKE USR "c"+3,BIN 01000000
700 POKE USR "c"+4,BIN 00111111
710 POKE USR "c"+5,BIN 00100100
720 POKE USR "c"+6,BIN 000110000
730 POKE USR "c"+7,BIN 00100100
740 POKE USR "d",BIN 01111100
750 POKE USR "d"+1,BIN 10000010
760 POKE USR "d"+2,BIN 00111010
770 POKE USR "d"+3,BIN 00000010
780 POKE USR "d"+4,BIN 11111100
790 POKE USR "d"+5,BIN 00100100
800 POKE USR "d"+6,BIN 00011000
810 POKE USR "d"+7,BIN 00100100
820 RETURN
```

Fig 8.9 Program to draw alien invader figure

Chapter Nine

Adding Depth and Perspective

The graphs and charts which we have drawn so far have had just two variables, X and Y, which were plotted horizontally and vertically on the screen. In the real world, however, there will be many situations where three variables are involved. The third variable is usually given the name Z. An example of this would be where we want to show the height of various points in a small area of land. In this case our X and Y co-ordinates would represent, say, length and width and would locate a particular point on the surface of the land. The third term Z will now be the height of the land surface at that pont. Here the value of Z will depend upon both X and Y since a change in either X or Y will take us to another point on the land surface with a different value for Z.

Drawing a three axis graph requires some slightly different techniques since we have to find a way of fitting in the Z axis. If X and Y are plotted as usual on the screen the Z ordinates should theoretically be plotted out from the surface of the screen. This is obviously impractical so we need to look at other possible schemes. One possibility is to plot Z against X on the screen for different values of Y to give a series of graphs, one for each value of Y. If these are plotted on top of one another the result would be rather confusing. A different colour could be used to draw each graph but this is not very satisfactory.

Suppose we were building a cardboard model of the three axis plot. The first step would be to plot a series of graphs of Z against X. Once the graphs had been plotted the next step might be to stand the graphs one behind the other. How can this be done in our display screen?

One solution to displaying such a series of graphs might be to draw them so that the graph for each new value of Y is displaced to the left and up on the screen. This helps to separate the individual graphs for the different values of Y. In effect we are now drawing the

Y axis along a sloping line which runs upwards and to the left of the X,Y,Z origin point where X, Y and Z are all zero. This is an improvement on superimposed graphs but still not quite right.

The usual solution to displaying a three axis plot is to draw both the X and Y axes at about 30 degrees to the horizontal axis of the screen Z axis vertical as shown in Fig. 9.1. Now as X increases the plotted point moves upwards and to the right whilst as Y increases the plotted point moves upwards and to the left. Finally Z just displaces the point vertically on the screen.

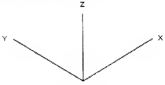


Fig 9.1. Layout of the X, Y and Z axes

Three axis bar charts

One type of display that looks impressive in a three axis version is a bar chart. The first step in constructing such a chart is to choose an origin point where the values of X, Y and Z are all at zero. This point determines where the bar chart will be displayed and also acts a reference point around which the plot will be constructed. The next step might be to draw a grid showing the X and Y co-ordinates in the plane where Z = 0.

To draw the X axis at an angle we need a Y movement which is half the X movement so our screen co-ordinates for points along the X axis (Y and Z both = 0) will be:

X1 = CX = X
Y1 = CY = X/2

When X and Z are both at 0 the line representing the Y axis must go up and to the left. Since the movement is to the left of the origin point (CX,CY) this means that the screen X co-ordinate for points along the Y axis must be less than CX. To get the 30 degree angle to the left, the X movement is made the same as the Y movement, but negative and the Y movement is halved. The screen co-ordinates here become

$$X1 = CX - X$$
$$Y1 = CY + X/2$$

When X and Z are both at 0 the line representing the Y axis must go up and to the left. Since the movement is to the left of the origin

```
100 REM 3 axis graph for Z=0
110 CLS
115 REM Set origin point
120 LET cx=116: LET cy=20
125 REM Set X and Y max values
130 LET xm=96: LET ym=80
135 REM Set X and Y steps
140 LET xs=12: LET ys=8
145 REM Draw X axis axis
150 FOR y=0 TO ym STEP ys
160 LET x1=-y
170 LET x2=xm-y
180 LET y1=y/2
190 LET y2=(xm+y)/2
200 PLOT INT (cx+x1),INT (cy+y1)
210 DRAW INT (x2-x1),INT (y2-y1)
220 NEXT y
225 REM Draw Y axis lines
230 FOR x=0 TO xm STEP xs
240 LET x1=x
250 LET x2=x-ym
260 LET y1=x/2
270 LET y2=(x+ym)/2
280 PLOT INT (cx+x1),INT (cy+y1)
290 DRAW INT (x2-x1),INT (y2-y1)
300 NEXT x
305 REM Insert scale markings
310 PLOT cx,cy
320 DRAW INT (xm+xs),INT ((xm+xs)/2)
330 PLOT cx,cy
340 DRAW -INT (ym+ys),INT ((ym+ys)/2)
350 PRINT AT 13,3;"Y";
360 PRINT AT 12,28;"X";
370 PRINT AT 20,14;"0";
```

Fig 9.2 The XY plane for Z=0

point (CX,CY) this means that the screen X co-ordinate for points along the Y axis must be less than CX. To get the 30 degree angle to the left, the X movement is made the same as the Y movement, but negative and the Y movement is halved. The screen co-ordinates here become:

X1 = CX − Y
Y1 = CY + Y/2

For any other point on the Z = 0 plane then the position of X1,Y1 will be produced by combining the two results we obtained above to give:

X1 = CX+X−Y
Y1 = CY+X/2+Y/2

At this point we can turn this into a piece of program, listed in Fig. 9.2, which produces a picture of the X,Y plane of the graph when Z =∅ as shown in Fig. 9.3.

The Z term is plotted vertically so it will only affect the Y value of a point on the chart. Since we are going to draw a vertical line to represent the Z ordinate we need to know the co-ordinates for the top of the line. Now the X value is the same as X1 and for the new Y value Z is simply added to Y so the values for co-ordinates X2,Y2 become:

X2 = X1 = CX+X−Y
Y2 = Y1+Z = CY+X/2+Y/2+Z

The Z ordinates can now be produced by drawing lines starting at X1,Y1 and running to point X2,Y2 by using a PLOT command to get to X1,Y1 and a DRAW to produce the line. As an example, let us assume that we wish to draw a graph for $Z = (X^2)/3+(y^2)/2$. To plot each Z point we place a vertical line whose length represents Z with its base at the required point in the XY plane. A program to draw such a graph is shown in Fig. 9.4.

The result is now a pattern of vertical lines looking like a bed of nails where the height of each line indicates the value of Z. To avoid the Z co-ordinates being merged together the steps on the X and Y axes must be different. The result on the screen is as shown in Fig. 9.5.

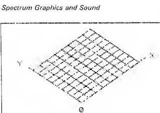


Fig 9.3 Display of X,Y plane

```
100 REM Simple 3 axis graph
110 CLS
115 REM Set origin point
120 LET cx=116: LET cy=20
125 REM Set X and Y max values
130 LET xm=96: LET ym=80
135 REM Set X and Y steps
140 LET xs=16: LET ys=10
150 DIM z(7,9)
155 REM Calculate and plot Z values
160 FOR y=1 TO 9
170 FOR x=1 TO 7
180 LET x1=x-1: LET y1=y-1
190 LET z(x,y)=x1*x1/3+y1*y1/2
200 LET x2=INT (xs*x1-ys*y1)
210 LET y2=INT ((xs*x1+ys*y1)/2)
220 PLOT cx+x2,cy+y2
230 DRAW 0,z(x,y)
240 NEXT x
250 NEXT y
255 REM Insert scale marks
260 PLOT cx,cy
270 DRAW INT (xm+xs),INT ((xm+xs)/2)
280 PLOT cx,cy
290 DRAW INT -(ym+ys),INT ((ym+ys)/2)
295 REM Insert scale markings
300 PRINT AT 13,2;"Y";
310 PRINT AT 12,29;"X";
320 PRINT AT 20,14;"0";
330 PLOT cx,cy
340 STOP
350 PRINT AT 13,3;"Y";
360 PRINT AT 12,28;"X";
370 PRINT AT 20,14;"0";
```

Fig 9.4 Simple three axis bar plot

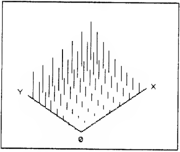


Fig 9 5 Display of simple three axis chart

Producing wide bars

The three axis bar chart we have produced so far has a simple line for each Z ordinate It can be made to look more attractive by turning the simple vertical line into a bar aligned along say the X axis The changes to the program are quite simple since they involve drawing another Z ordinate of the same height but at a slightly different position and then linking the top and bottom of these two ordinates with a short line The area within the box so produced is then filled with INK colour Taking the same set of Z values as before the revised program becomes as shown in Fig 9 6

The bars are spaced equally apart with the space between them equal to the bar width The top and bottom of the bars are drawn parallel to the X axis Note that both X and Y are decremented from maximum to zero so that bars at the rear of the graph are drawn first After it has been filled with colour each bar is outlined in the background colour by using the INVERSE 1 command before drawing the outline of the bar After the bar has been outlined in PAPER colour INVERSE ∅ is used to restore the normal plotting and drawing mode This makes the bars in front stand out where they overlap another bar as shown in Fig. 9 7.

```
100 REM 3 axis chart with wide bars
110 REM z=(4+2*SIN (x/20))*(y/10+1)
120 LET cx=128: LET cy=16
130 LET dx=10: LET dy=5
135 REM Draw x,y grid
140 GO SUB 400
145 REM Draw chart
150 FOR x=100 TO 0 STEP -20
160 FOR y=90 TO 0 STEP -15
170 LET z=INT ((4+2*SIN (x/20))*(y/10+1))
180 GO SUB 500
190 NEXT y
200 NEXT x
210 STOP
390 REM x,y grid subroutine
400 FOR x=0 TO 100 STEP 20
410 PLOT cx+x,cy+x/2
420 DRAW -100,50
430 NEXT x
440 FOR y=0 TO 90 STEP 15
450 PLOT cx-y,cy+y/2
460 DRAW 110,55
470 NEXT y
480 RETURN
500 REM Bar drawing subroutine
510 FOR n=0 TO z-1
520 PLOT cx+x-y,cy+x/2+y/2+n
530 DRAW dx,dy
540 NEXT n
545 REM Erase bar outline
550 INVERSE 1
560 PLOT cx+x-y,cy+x/2+y/2
570 DRAW dx,dy
580 DRAW 0,z
590 DRAW -dx,-dy
600 DRAW 0,-z
610 INVERSE 0
620 RETURN
```

Fig 9.6 Three axis chart with wide bars

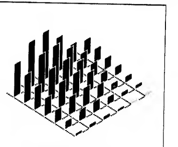


Fig 9.7 Wide bar chart screen display

Producing solid bars

A further development is to draw the bars so that they appear to be solid. In effect we draw one bar aligned along the X axis and another aligned with the Y axis for each Z ordinate. Finally a diamond shaped top is drawn to complete the bar. One side of the bar may then be filled with colour as desired.

```
100 REM 3 axis chart with solid bars
110 REM z=(4+2*COS (x/20))*(y/10+1)
120 LET cx=128: LET cy=16
130 LET dx1=10: LET dy1=5
140 LET dx2=8: LET dy2=4
150 REM Draw x,y grid
160 GO SUB 400
165 REM Draw chart
170 FOR x=100 TO 0 STEP -20
180 FOR y=90 TO 0 STEP -15
190 LET z=INT ((4+2*COS (x/20))*(y/10+1))
195 REM Draw bar
200 GO SUB 500
210 NEXT y
```

```
220 NEXT x
230 STOP
390 REM x,y grid subroutine
400 FOR x=0 TO 100 STEP 20
410 PLOT cx+x,cy+x/2
420 DRAW -100,50
430 NEXT x
440 FOR y=0 TO 90 STEP 15
450 PLOT cx-y,cy+y/2
460 DRAW 110,55
470 NEXT y
480 RETURN
500 REM Bar drawing subroutine
505 REM Draw face of bar
510 FOR n=0 TO z-1
520 PLOT cx+x-y,cy+x/2+y/2+n
530 DRAW dx1,dy1
540 NEXT n
545 REM Erase side of bar
550 INVERSE 1
560 FOR n=0 TO z-1
570 PLOT cx+x-y,cy+x/2+y/2+n
580 DRAW -dx2,dy2
590 NEXT n
600 INVERSE 0
605 REM Draw side of bar
610 PLOT cx+x-y,cy+x/2+y/2
620 DRAW -dx2,dy2
630 DRAW 0,z
640 DRAW dx2,-dy2
650 DRAW 0,-z
660 REM Erase top of bar
670 INVERSE 1
680 FOR n=0 TO dx2-1
690 PLOT cx+x-y-n,cy+z+(x+y+n)/2
700 DRAW dx1,dy1
710 NEXT n
720 INVERSE 0
725 REM Draw top of bar
730 PLOT cx+x-y,cy+x/2+y/2+z
740 DRAW dx1,dy1
750 DRAW -dx2,dy2
760 DRAW -dx1,-dy1
770 DRAW dx2,-dy2
780 RETURN
```

Fig 9.8 Three axis bar chart with solid bars

The program listing shown in Fig. 9.8 produces an example of this type of display and the result on the screen is shown in Fig. 9.9. In the program the front face of the bar is drawn first and completely filled with INK colour. Next the side of the bar along the Y axis is drawn using INVERSE 1 which effectively erases any other bars that lie behind the one being drawn. INVERSE Ø is then used to restore normal drawing and the outline of the side of the bar is drawn. Finally the diamond shaped top of the bar is erased using INVERSE and then its outline is drawn.

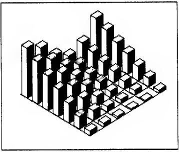


Fig. 9.9 Display of 3 axis chart with solid bars

X—Y bar chart with solid bars

A simple X,Y two-dimensional chart could also be drawn with pseudo solid bars. In this case the X axis is tilted to 30 degrees and the background to the bars is drawn in first. The bars are then drawn on top using say half the X step for bar thickness and width. The technique for drawing the bars here is exactly the same as that used for the three axis chart with solid bars.

A program to produce this type of display is shown in Fig. 9.10. The result produced on the screen is shown in Fig. 9.11.

```
100 REM Bar chart with solid bars
110 REM for y=x*x/100
120 LET cx=64: LET cy=16
130 LET ax=6: LET ay=3
140 LET bx=8: LET by=4
150 LET lx=100: LET ly=100
160 REM Draw background plane
170 GO SUB 400
180 REM Draw chart
190 FOR x=100 TO 0 STEP -15
200 LET y=x*x/100
205 REM Draw bar
210 GO SUB 500
220 NEXT x
230 STOP
390 REM Background subroutine
400 PLOT cx-bx,cy+by
410 DRAW lx,lx/2
420 DRAW 0,ly
430 DRAW -lx,-lx/2
440 DRAW 0,-ly
450 RETURN
500 REM Bar drawing subroutine
505 REM Draw face of bar
510 FOR n=0 TO y-1
520 PLOT cx+x,cy+x/2+n
530 DRAW ax,ay
540 NEXT n
545 REM Erase side of bar
550 INVERSE 1
560 FOR n=0 TO y-1
570 PLOT cx+x,cy+x/2+n
580 DRAW -bx,by
590 NEXT n
600 INVERSE 0
605 REM Draw side of bar
610 PLOT cx+x,cy+x/2
620 DRAW -bx,by
630 DRAW 0,y
640 DRAW bx,-by
650 DRAW 0,-y
660 REM Erase top of bar
670 INVERSE 1
680 FOR n=0 TO bx-1
690 PLOT cx+x-n,cy+y+x/2+n/2
```

```
700 DRAW ax,ay
710 NEXT n
720 INVERSE 0
725 REM Draw top of bar
730 PLOT cx+x,cy+x/2+y
740 DRAW ax,ay
750 DRAW -bx,by
760 DRAW -ax,-ay
770 DRAW bx,-by
780 RETURN
```

Fig 9.10 Two axis chart with solid bars

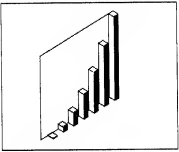


Fig 9.11 Display of 2 axis chart with solid bars

Surface Maps

Instead of drawing vertical lines for the Z ordinates we could draw a picture showing the contours of the surface made up by the tips of the Z ordinates. Here we would take all Z ordinates with Y=0 and draw lines linking them together. Then we repeat the process for each value of Y to give a series of lines running diagonally up to the left. Next we take all Z ordinates for X=0 and again link them with lines. This is repeated for all X co-ordinates and finally we have a patchwork pattern which will give an impression of the way that Z varies over the X,Y plane. Again, we might use colour to pick out the different strips across the plane.

The results obtained using either the vertical Z ordinates or the linked points on the surface will be best if there are a large number of points since this gives a smoother contour to the figure. However if too many points are used the solid image effect will tend to be lost as some points will overwrite others and produce some confusion. The best results will generally be obtained by arranging the drawing sequence so that it starts at the points toward the back of the X,Y plane, that is with both X and Y at their highest values. Any lines that do overlap will now be overwritten by the line that should appear in front.

Circular three axis plots

A rather interesting variation of three axis plotting is the circular three axis plot which can produce some rather attractive patterns.

In this version of the three axis graph the X,Y plane of the chart is made elliptical so that the resultant plot is displayed on the screen as a sort of disc viewed from an angle with elliptical ridges produced by the Z ordinates.

The technique of plotting this type of graph makes use of the quadratic method for drawing a circle to produce the X,Y axes. The X scale is set at perhaps two or three times the Y scale to produce an

```
100 REM Circular 3 D graph
110 REM Set up function
120 LET k=PI/2000
130 LET m=1/SQR (2)
140 DEF FN a(z)=10*COS (k*(x*x+y*y))
150 REM Plot graph
160 FOR x=-100 TO 100
170 LET y1=5*INT (SQR (10000-x*x)/5)
180 FOR y=y1 TO -y1 STEP -5
190 LET z=FN a(SQR (x*x+y*y))-m*y
195 REM Hidden line removal
200 IF y=y1 THEN  GO TO 220
210 IF z<z1 THEN  GO TO 240
220 PLOT 128+x,80+INT (z/2)
230 LET z1=z
240 NEXT y
250 NEXT x
```

Fig 9.12 Circular 3-D type plot

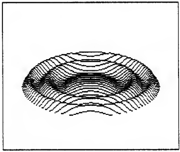


Fig. 9.13. Display of circular three axis chart

elliptical figure on the screen. Z values are simply added to the calculated Y co-ordinates and points are plotted at the tops of the Z ordinates.

The program of Fig. 9.12 gives an example of this type of plot. The function used will determine the contour shape of the display and you could try a variety of functions here. A typical display appears like Fig. 9.13. Note that this type of display takes quite a long time to generate because of the large number of points and the relatively complex calculations for each point.

Perspective drawings

The three axis charts or graphs which we have produced so far do give some impression of depth but to an artist they will look all wrong. These are, in fact, what are known as *isometric* drawings. This basically means that a vertical line always has the same scale factor wherever it is on the X Y plane. This type of drawing is often produced by draughtsmen because it allows correct measurements to be made of the object along all three axes.

To create the illusion of depth an artist uses what is known as *perspective* in his drawings. Artists had discovered many centuries ago the effect that as an object is moved further away from the viewer it appears to get smaller and vice versa. They also found that by applying this idea to drawings and paintings they could produce a much more realistic picture. This technique is known as perspective drawing and over the years mathematicians have evolved formulae to allow the shape and size of objects to be calculated to give a correct perspective view. This technique can also be applied in computer graphics to produce more realistic displays.

To see how perspective works, imagine that you are standing on a flat plain and that in front of you is a road that stretches away to the horizon. Although the sides of the road are actually parallel it will appear that the road gets narrower as it approaches the horizon. The cars and trucks travelling along the road also appear to get smaller as they move away from your position toward the horizon. In fact the optical image they produce does get smaller as they move away. If we apply this basic rule to our pictures on the screen we can also produce an illusion of depth despite the fact that our display is really a flat screen.

Firstly we need to decide on some system of co-ordinates by which we can measure the positions of points on the objects being viewed and the corresponding points needed to produce the screen image. We shall assume that the X axis runs across from left to right as usual. The Z axis is normally the vertical direction as we had it in our three axis graphs. This leaves the Y axis and the best arrangement is to have the Y axis along the direction of view.

If you viewed the road across the desert from actual road level (that is, with your eye actually at the road surface) the view would be rather uninspiring because every point on the road and the desert would lie along a single line through the X axis. In order to see the road properly we need to be located above it.

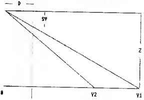


Fig 9.14 Diagram showing relationship of road and screen.

Figure 9.14 shows a side view of the situation where we are viewing the road from an altitude Z. In order to project the image on our flat screen we shall assume that we are looking through a window at distance D.

Suppose we take a point on the road at distance Y1. This will appear to be below the horizon line on the screen by an amount SY. We have assumed here that the horizon is effectively at eye level which it will be if we are looking along a line parallel to the Y axis. Now the small triangle between the screen and eye is of the same shape as the large triangle passing through point Y1. This means that all of the sides of the two triangles have the same proportions so we can say that:

SY/D = Z/Y1

and rearranging this we get:

SY = Z*D/Y1

Now you will note that the size of the image on the screen is inversely proportional to the distance Y1. If we took another point on the road at a different distance Y2 then it would produce a different line length on the screen giving a new value for SY of:

SY = Z*D/Y2

Now suppose there were a series of equal length lines drawn along the centre line of the road. Each line will produce a short vertical line on the screen and as the point on the road gets farther away the image produced on the screen by each line gets shorter.

If we drew a similar view of the road looking down on it we would find that the same basic formula applies for the SX image size on the screen which represents the road width. As Y1 becomes greater the width of the image on the screen decreases according to the formula:

SX = W*D/Y1

where W is the width of the road.

Vertical objects alongside the road will also produce images that follow this general rule of being inversely proportional to distance Y.

Let us start with the road. Firstly we need to draw a horizon line horizontally across the screen since this acts as a visual reference. Now if we draw lines from the bottom two corners of the screen to a point halfway across the horizon line we have a road.

Suppose the road is 25 feet wide and we are viewing a 14 inch TV screen from, say, a distance of 4 feet. The TV screen will be 1 foot wide and this is equal to SX. D will be 4 and X will be 25. Now

SX = D*Y/Y = 4*25/Y = 1

therefore

Y = 4*25 or 1ØØ feet

The scaling factor for the X direction in our screen drawing can now be worked out. There are 256 units of X across the screen on the Spectrum and when Y=100 the road fills the screen width, so SX must be 256. If we rewrite out equation with a multiplier SCX we get

SX = SCX × X/Y

and inserting our calculated values we get the following:

256 = SCX × 25/100

now extracting the SCX term we get

SCX = 256 × 100/25 = 1024

To calculate values of SX we would use the equation

SX = 1024 * X/Y

If we apply a similar process to calculate the Y scale factor, assuming that our viewpoint is at a height of 10 feet and that for a distance of 100 feet the point plotted is at the bottom of the screen. We shall assume that the horizon line is at Y = 96 on the screen. From this

SY = SCY × 10/100 = 96

and

SCY = 960

so to calculate the Y points on the screen we would use

SY = 960*Z/Y

To plot a point on the road itself Z=1Ø and the actual Y value for use in the drawing instruction will be 96−SY since as we get closer the point moves down the screen. If there is a vertical pole then to plot the top of the pole we subtract the height of the pole from 10 to obtain the value for Z. Thus a 10 foot high pole will always produce a Y value of 96 and would line up with the horizon since we are actually looking along a line 10 feet above the road. If the pole is higher than 10 feet the value of SY is negative and the top of the pole goes above the centre line of the screen.

To draw a perspective view of a road with, say, trees alongside you will need to set up values of height for the tree trunk an the top of the tree and also the width of the tree. Knowing the distance of each tree from the viewer, its X,Y co-ordinates for the base, top and side points can be calculated using:

$$SX = 128 + 1024 * X / Y$$

$$SY = 96 - 960 * (10 - Z) / Y$$

where SX and SY are the screen drawing co-ordinates, X is the distance measured from the centre of the road with positive values to the right and Z is the height of the object. Once the co-ordinates are known the tree can be drawn using a series of PLOT and DRAW commands. The road markings are dealt with in a similar fashion except that here Z will be ϕ.

Figure 9.15 shows a program listing to draw a perspective view of a road and Fig. 9.16 shows the result on the screen.

```
100 REM Perspective view of road
105 REM Draw sky
110 PAPER 5: CLS
115 REM Draw desert
130 PAPER 6
140 FOR y=10 TO 21
150 FOR x=0 TO 31
160 PRINT AT y,x;" ";
170 NEXT x: NEXT y
175 REM Draw road
180 FOR x=-128 TO 127
190 PLOT  INK 0;128,96
200 DRAW  INK 0;x,-96
210 NEXT x
220 LET sx=1024: LET sy=960
225 REM Draw road markings
230 FOR y=100 TO 2000 STEP 100
240 LET y1=INT (sy*10/(y+50))
250 LET y2=INT (sy*10/y)
260 LET x=INT (sx*1/(y+50))
270 FOR k=128-x TO 128+x
280 PLOT  PAPER 7; INVERSE 1;k,96-y1
290 DRAW  PAPER 7; INVERSE 1;0,y1-y2
300 NEXT k
```

```
310 LET x1=INT (sx*1/y)
320 FOR k=0 TO x1-x
330 PLOT  PAPER 7; INVERSE 1;128-x,96-y1
340 DRAW  PAPER 7; INVERSE 1;-k,y1-y2
350 PLOT  PAPER 7; INVERSE 1;128+x,96-y1
360 DRAW  PAPER 7; INVERSE 1;k,y1-y2
370 NEXT k: NEXT y: INVERSE 0
375 REM Draw horizon
380 PLOT  INK 0;0,96
390 DRAW  INK 0;255,0
395 REM Draw trees
400 LET tr=5: LET tt=20: LET tw=3
410 FOR y=150 TO 1200 STEP 150
420 LET y1=INT (sy*10/y)
430 LET y2=INT (sy*(10-tr)/y)
440 LET y3=INT (sy*(10-tt)/y)
450 LET x=INT (sx*12.5/y)
460 LET x1=INT (sx*tw/y)
470 LET x2=128-x: GO SUB 1000
480 LET x2=128+x: GO SUB 1000
490 NEXT y
500 STOP
990 REM Tree drawing subroutine
1000 PLOT  INK 0;x2,96-y1
1010 DRAW  INK 0;0,y2
1040 FOR k=-x1 TO x1
1050 PLOT  INK 0;x2,96-y3
1060 DRAW  INK 0;k,y3-y2
1070 NEXT k: OVER 0
1080 RETURN
```

Fig 9.15 Program to draw a road

Wire frame models of solid objects

When we come to drawing perspective images of three-dimensional objects we use an array of X, Y, Z co-ordinates to define points on the object. When the image is drawn these points are linked together by lines to produce an outline of the edges of the object. If we were drawing a rectangular block then the reference points would be the eight corners of the block and the linking lines represent the edges of the block. This form of drawing is called a *wire frame image* because it is as if we made up the object by building a wire frame marking out its edges.

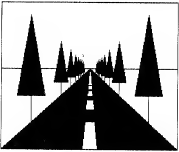


Fig 9.16 Picture of road

For viewing the object along its Y axis we can use the same basic proportional technique that was used for the perspective view of a road. If we want to be able to move all around our object so that we can view it from all directions then the equations become more complex. We shall now go on to look at this situation.

All round viewing of objects

To produce a perspective view of a wire frame object from any point around the object becomes fairly complex. Firstly we need to define the object as a set of X,Y,Z co-ordinates relative to its own centre. We shall also need a set of information which shows where the edges are relative to the X, Y, Z points. In addition we will need some information to tell the computer whether it has to draw a line or not when tracing out the image from point to point on the screen. This can be arranged by creating three arrays of data describing the object. The X, Y and Z co-ordinates of the object in its descriptive array are all measured relative to a point at the centre of the object itself. Next we have a viewer at some point XV, YV, ZV relative to the object.

The first stage in the process is to translate the X, Y, Z co-ordinates of the object so that the viewer is placed at the origin of the X, Y, Z axes. This is the point where X, Y and Z are all 0. This translation process is quite straightforward since it simply involves subtracting XV,YV,ZX from X,Y and Z respectively to give a new set of values for the X,Y and Z co-ordinates of the points on the object.

At this point we have assumed that the viewer is absolutely level. For all round viewing we can assume that the line of sight of the viewer has three angular components which we shall call heading, pitch and roll. To understand these it is useful to imagine that you are flying in an aeroplane. The Y axis of the aeroplane is assumed to be along the fuselage and the X axis along the wings.

Heading is the direction of view of the viewer measured relative to the Y axis. A change in heading is equivalent to turning to the left or right. This is effectively a rotation of the aircraft around its Z or vertical axis. Next there is pitch which shows whether you are looking above or below the horizontal. This shows whether the aircraft has a nose down or nose up attitude and is equivalent to rotating the plane around the line along its wings. This is effectively a rotation about the X axis of the aircraft. Finally there is roll which indicates if your view is inclined to the right or left. This is like rotating the plane around a line along its fuselage which is what happens when an actual aircraft rolls. Here the rotation is around the Y axis of the aircraft.

After shifting the origin of the X, Y, Z map we will have the poition where the viewer and the object being viewed are on the Y axis, but the viewer is in fact looking at a point away from the Y axis by his heading angle. This could place the object off the screen. In order to place the object in the centre of the field of view and at its correct orientation we will need to rotate its points about the three axes to correct for the heading, pitch and roll of the viewer. This is done in three stages.

First the points are rotated around the Z axis until the viewer is looking directly through the centre of the object. This is done by rotating all of the points on the object through the heading angle of the viewer. The rotation equations are the same as those used in Chapter Four but now they are applied to X,Y and Z. In this first rotation, Z1 will be equal to the original Z value since we are rotating around the Z axis itself. The new X and Y values become:

```
X1=X*COS(H)-Y*SIN(H)
Y1=X*SIN(H)+Y*COS(H)
```

In the second stage of calculations these X1,Y1,Z1 values are rotated by the pitch angle to produce a new set of values X2,Y2,Z2. In this rotation the X values are unchanged since the points are being rotated around the X axis. The second set of values X2, Y2, Z2 are finally given a further rotation by the roll angle to produce the co-ordinates X3,Y3,Z3. In this last rotation which is around the Y axis the Y2 values will be unchanged.

Having produced these rotated co-ordinates we have reached roughly the position we were in with the road view. In remains to project the points on to the screen and this basically involves dividing X3 and Z3 respetively by Y3 to obtain the screen co-ordinates XS and YS.

This whole process of rotation and projection is carried out for each co-ordinate point on the object and then the appropriate lines are plotted between the points to form the picture on the screen.

The program listing given in Fig. 9.17 carries out this process on an object which is a wire frame model of a simple block. The front

```
100 REM All round perspective
110 REM view of a block
120 DIM v(50,3): DIM e(100): DIM l(100)
130 LET sx=10: LET sy=10
135 REM Set up data on figure
140 READ nv
150 FOR p=1 TO nv
160 READ v(p,1),v(p,2),v(p,3)
170 NEXT p
180 READ ne
190 FOR j=1 TO ne
200 READ e(j),l(j)
210 NEXT j
220 REM Set viewer position
230 LET d=80: LET p=22
240 LET r=0: LET h=45
245 REM Main program loop
250 CLS : PRINT "Heading= ";h
260 PRINT "Pitch= ";p
270 PRINT "Roll= ";r
280 LET h=h*PI/180
290 LET p=p*PI/180
300 LET r=r*PI/180
305 REM Calculate multipliers
310 GO SUB 1000
320 LET xv=-d*cp*sh
```

```
330 LET yv=-d*cp*ch
340 LET zv=-d*sp
350 REM Project image on screen
360 LET x1=0: LET y1=0
370 FOR j=1 TO ne
380 LET n=e(j)
390 LET x=v(n,1): LET y=v(n,2)
400 LET z=v(n,3)
410 GO SUB 1200
415 REM Check if line to be drawn
420 IF l(j)=0 THEN  GO TO 450
425 REM Draw line
430 PLOT x1,y1
440 DRAW xs-x1,ys-y1
445 REM Update cursor position
450 LET x1=xs: LET y1=ys
460 NEXT j
480 INPUT "Another view (y/n)";a$
490 IF a$<>"y" THEN  STOP
500 INPUT "Heading (deg)=";h
510 INPUT "Pitch (deg)=";p
520 INPUT "Roll (deg)=";r
530 GO TO 250
540 STOP
1000 REM Multiplier factors
1010 LET ch=COS h: LET sh=SIN h
1020 LET cp=COS p: LET sp=SIN p
1030 LET cr=COS r: LET sr=SIN r
1040 LET m1=ch*cr-sh*sp*sr
1050 LET m2=-sh*cr-ch*sp*sr
1060 LET m3=cp*sr
1070 LET m4=sh*cp
1080 LET m5=ch*cp
1090 LET m6=sp
1100 LET m7=ch*sr-sh*sp*cr
1110 LET m8=-sh*sr-ch*sp*cr
1120 LET m9=cp*cr
1130 RETURN
1190 REM
1200 REM Move viewer position
1210 LET x=x-xv: LET y=y-yv: LET z=z-zv
1220 REM Rotate view
1230 LET x3=m1*x+m2*y+m3*z
1240 LET y3=m4*x+m5*y+m6*z
```

```
1250 LET z3=m7*x+m8*y+m9*z
1260 REM Calculate x,y position on screen
1270 LET xs=128+INT (sx*d*x3/y3)
1280 LET ys=80+INT (sy*d*z3/y3)
1290 RETURN
1990 REM
2000 REM Number of points
2010 DATA 12
2020 REM x,y,z coordinates
2030 DATA 5,-3,4
2040 DATA -5,-3,4
2050 DATA -5,-3,-4
2060 DATA 5,-3,-4
2070 DATA 5,3,4
2080 DATA -5,3,4
2090 DATA -5,3,-4
2100 DATA 5,3,-4
2110 DATA 4,-3,3
2120 DATA -4,-3,-3
2130 DATA 4,-3,-3
2140 DATA -4,-3,3
2199 REM
2200 REM Number of edges
2210 DATA 20
2220 REM Edge and line data
2230 DATA 1,0,2,1,3,1,4,1
2240 DATA 1,1,5,1,6,1,7,1
2250 DATA 8,1,5,1,2,0,6,1
2260 DATA 3,0,7,1,4,0,8,1
2270 DATA 9,0,10,1,11,0,12,1
```

Fig. 9.17 Program to view a 3-D block

face of the block has a cross drawn on it to make it a little easier to interpret the picture. By inputting the direction of view in terms of heading, pitch and roll, as seen from the viewer's position, the corresponding view of the object is displayed. In effect we are rotating the object itself to present the correct view. Fig. 9.18 shows a typical view on the screen.

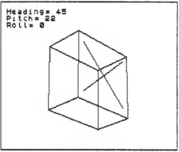


Fig 9 18 Typical display of the block

You can produce a different shape by setting up new data arrays. The x,y,z co-ordinates define the corners of the object. The edge data shows the sequence of coordinates through which lines are to be drawn. They are in pairs the first item being the coordinate number and the second is a 1 if a line is to be drawn or a 0 if no line is required. Remember to change the data values specifying the number of points and edges.

Chapter Ten
Making Sounds and Music

One aspect of home computers that seems to have become more important in recent years is the production of sounds and music. This is particularly important where the computer is used as a games machine, as a visit to any video game arcade will show.

Some modern home computers have very advanced sound producing systems with perhaps three or four independent sound channels and full control over the volume, frequency and duration of the sounds produced. Such machines are capable of producing an almost infinite variety of sounds. The Spectrum has a much more modest sound capability with only one channel and only one BASIC command for the control of sound generation.

Like many other small home computers, the Spectrum has a loudspeaker built into the computer case for producing sound. Inevitably this has to be a small loudspeaker in order to fit it into the Spectrum and such small loudspeakers cannot give particularly good sound reproduction. A further limitation is that the volume of sound produced tends to be rather low. It is possible however to take a sound output signal from the MIC socket which is normally used to drive the cassette recorder when storing programs. If the signal from the MIC socket is fed to a hi-fi amplifier much better sound outputs can be produced from the Spectrum. There are several small add on amplifier units available for the Spectrum and these also use the MIC signal to provide a louder sound output.

What is sound?

All of the sounds that we normally hear are produced by pressure waves in the air around us. To see how this works imagine a stone thrown into a pond. When the stone hits the water it produces a series of ripples in the water surface which spread outwards from the point where the stone entered. Sound waves are similar to those ripples on the water except that in the case of sound waves the ripples consist of changes in the air pressure which radiate from the source of the sound. As the sound wave passes us the air in contact with our ears is alternately compressed and expanded in sympathy with the sound wave. Inside the ear the vibrations produced by the sound wave are converted into nerve impulses and we sense the sound.

A pure sound tone will have a pressure wave with a sinusoidal waveform as shown in Fig. 10.1. The height or amplitude of the wave determines how loud the sound is. This is often referred to as the *volume* of the sound. In the Spectrum there is no direct means of controlling the sound volume so all sounds produced have roughly the same loudness.

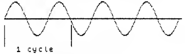


Fig. 10.1. Sinusoidal wave of sound.

The rate at which the sine wave changes determines the *pitch* or *frequency* of the sound. Thus the faster the wave changes, the higher will be the pitch. The frequency or pitch of a sound is measured as the number of complete cycles of the sine wave that occur in one second and is generally quoted in units called hertz (Hz). For one cycle the sound signal starts at zero, rises to its maximum positive value, passes through zero to its maximum negative value and finally returns to zero again as shown in Fig. 10.1. Thus a sound wave which passes through 50 cycles every second would be said to have a frequency of 50 Hz. This sound would be a low pitched hum and is the sound that will often be heard coming from electrical equipment since it is the frequency of the mains electricity supply.

The typical range of sounds that can be detected by the human ear is from about 40 Hz up to around 15000 Hz (15 kHz). In the Spectrum the frequency of the sounds that can be produced ranges from about 10 Hz at the low frequency end up to about 15000 Hz at the high end.

Apart from frequency and volume, all sounds will also have a *duration*, which is the length of time for which the sound is generated. The Spectrum allows the duration of the sound produced to be controlled by the program.

A sound signal with a sinusoidal waveform produces a pure note similar to that produced by a flute. Changing the shape of the wave to a square wave as shown in Fig. 10.2 produces a richer sound. Most computers tend to produce these square wave sound signals since they are much easier to generate electronically.

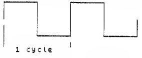


Fig. 10.2 A typical square wave signal

So far we have considered the sound wave as being of constant frequency and volume throughout its duration. In real life the characteristics of a sound are also greatly influenced by the way in which the amplitude and frequency vary as the sound is being produced. This is known as the sound *envelope*. With the Spectrum we cannot vary the amplitude of sounds so this restricts the types of sound that can be produced. We can, however, vary the frequency with time to produce various kinds of siren effect.

Sound generation in the Spectrum

In the more sophisticated computers special sound generator *chips* are generally used to produce the sound outputs under the control of the CPU chip inside the computer. Such a scheme usually allows the computer CPU to perform other tasks as the sound is being produced. In the Spectrum the sound is generated directly by the CPU itself. The basic technique is to switch a single output line on and off at regular intervals and to use the signal on this line to drive the loudspeaker. Each time the output signal is turned on and off it produces a 'click' in the loudspeaker. By producing a stream of successive clicks a rather rough audio tone is produced and the pitch or frequency of the tone will depend upon the rate at which the clicks are produced.

The duration of the tone produced from the Spectrum depends upon the length of the stream of clicks and this can readily be controlled by the program as can the pitch or frequency of the tone. One major disadvantage of this technique for producing sound is that whilst the Spectrum is generating a sound output the CPU is completely taken up by that task and all other computing activity ceases. This can of course present problems if you are trying to produce an animated graphics display at the same time as a sound effect. The problem can be overcome to some extent, however, by breaking up the sound signal into sections and interleaving the updating of the display with the sound commands.

The BEEP command

In the Spectrum the BASIC command used for producing sounds is called BEEP and takes the following form

```
1ØØ BEEP duration,pitch
```

The duration of the tone is measured in units of seconds so that if duration is set to 1 the tone will last for a period of 1 second. To see how this works try running the short program listed in Fig. 10.3. For most purposes the duration value will be in the range from Ø to 1.

```
100 REM Demonstration of the
110 REM duration of BEEP
120 LET d=0.01
130 FOR n=1 TO 10
140 FOR n=1 TO 10
150 PRINT AT 1,1;"Duration = ";d;" seconds       ";
160 BEEP d,0
170 PAUSE 25
180 LET d=d*2
190 NEXT n
```

Fig. 10.3 Demonstration of changes of duration of sound.

The pitch parameter is an integer number and must lie in the range from −59 to +69. A value of −59 gives a very low frequency buzzing or clicking sound and as the pitch numbers increase the pitch rises through the audible range and eventually becomes too high to be heard. At the low end the frequency is about 10 pulses per second and at the highest end rises to about 15000 cycles per second or 15 kilohertz (kHz). The note Middle C on a piano scale, which has a frequency of about 261 cycles per second, is produced by a pitch number P = ∅

To see the range of tones available from the Spectrum BEEP command try running the program shown in Fig 10.4. In this program the pitch is stepped through its entire range of values(−59 to 69) to give a sequence of tones rising in frequency. The duration is set at 0.5 to give half second long tones and a PAUSE 12 command after each note is used to separate the individual notes so that they can easily be picked out

```
100 REM Range of sound pitch
110 FOR p=-60 TO 69
120 PRINT AT 1,1;"Pitch = ";p;"      ";
130 BEEP 0.5,p
140 PAUSE 12
150 NEXT p
```

Fig. 10.4. Demonstration of range of tones available

The sounds which you can produce add considerable interest to most games programs played on your Spectrum.

Making sound effects

We can produce some simple sound effects by using the BEEP command. If the pitch is made to rise and fall regularly a siren type sound can be produced as shown by the little program listed in Fig. 10.5.

```
100 REM Siren type sound
110 FOR p=5 TO 15
120 BEEP .05,p: NEXT p
130 FOR p=15 TO 5 STEP -1
140 BEEP .05,p: NEXT p
150 GO TO 110
```

Fig. 10.5. Producing siren type sounds.

Other possibilities with the BEEP command include playing two sounds in rapid succession by interleaving two BEEP commands in a FOR ... NEXT loop and having a small difference in pitch between them as shown in Fig. 10.6.

```
100 REM Interleaved notes
110 FOR p=1 TO 4
120 FOR n=1 TO 50
130 BEEP .05,5
140 BEEP .05,5+p
150 NEXT n
160 PAUSE 25
170 NEXT p
```

Fig. 10.6 Beat frequency generation

This program produces a sort of warbling sound due to the beat between the two tones. Try different values for the pitch of the second sound to see the effects produced.

Making music

Since the Spectrum can produce a wide range of sound frequencies it can be used to play music. Of course the limitations of the sound generating system will not allow us to produce high quality sound but nevertheless tunes can be played on the Spectrum. These could be useful in games programs where different 'jingles' can be played when the player wins or loses.

When we consider the playing of a piece of music the sounds which make up the tune are generally called *notes*. Each note has a specific pitch or frequency which is related precisely to the pitch of the other notes in the musical scale. The complete pitch range used in music is divided up into groups of notes which are called *octaves*.

If we examine an octave of notes on say a piano keyboard as illustrated in Fig. 10.7, it consists of seven so called *natural* notes which are produced by the white piano keys. The natural notes are labelled A, B, C, D, E, F and G in order of ascending pitch. This sequence of notes from A to G is repeated up the scale so that the next note after G will be the A at the start of the next octave. This A is the eighth natural note above the next lower A in the set and it is from this that the name octave (eighth) is derived.

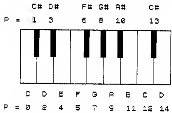


Fig. 10.7 The piano keyboard and note pitch numbers required

Between the natural notes there are some extra notes which are called *sharp* notes. The sharp notes are a semitone higher in pitch than the adjacent natural note. The sharp notes use the same letters as natural notes but the letter is followed by a crosshatch or hash (#) symbol. Thus the note C# is a semitone higher than C.

In music you will also find references to *flat* notes which are a semitone below a natural note. Sharps and flats are really just different ways of labelling the same note. If we consider the note A# which is a semitone above A, this will have the same pitch as the B flat note which is a semitone below B. In written music the flat is denoted by a symbol like a small b placed after the note.

You will see that there are only five sharp notes compared with seven natural notes in an octave. Most of the natural notes are separated from the next natural note by a sharp note, so that the pitch difference between adjacent natural notes is two semitones or a whole tone. The exceptions are B and E which have no corresponding sharp note so the pitch difference between B and C or between E and F is only a semitone. This rather odd arrangement seems to work quite well in practice and if the natural notes are played in succession we get the familiar musical scale.

Playing musical notes

Because of the twelfth root of two ratio between successive notes the actual frequencies of musical notes are all rather odd numbers. For instance the note called Middle C has a frequency of 261.7 Hz. The Middle C note is often used as a reference in computer sound systems and the Spectrum is no exception in this respect. In the Spectrum the BEEP frequency is actually specified as a number of semitones relative to Middle C so that if we use the command:

```
BEEP 1,0
```

the machine will produce a Middle C note for a period of one second. A positive pitch number will indicate the number of semitones above Middle C and a negative number indicates that the note is below Middle C.

To produce our familiar 'do ray me' type musical scale we can play the natural notes from Middle C up to and including the C in the next higher octave. Here we cannot use a simple counting loop, as we did to demonstrate the range of sounds, because the pitch numbers required are not all equally spaced. The correct scale can be played by setting up the required sequence of notes as an array and then repeating the BEEP command in a loop with varying pitch terms as shown in the program listed in Fig. 10.8.

```
100 REM Musical scale program
110 FOR k=1 TO 20
120 RESTORE
130 FOR n=1 TO 8
140 READ p
150 BEEP .25,p
160 NEXT n
170 PAUSE 25
180 NEXT k
190 DATA 0,2,4,5,7,9,11,12
```

Fig. 10.8 Playing a musical scale

Scales can also be played starting from a different note but to get the correct sequence of sounds this will involve using some of the sharp notes to form the scale.

Translating music

So far we have produced a musical scale and we could go on to play a tune by merely writing down the sequence of notes as letters and then converting them into pitch numbers for the BEEP command.

In practice, however, music is not normally written as a sequence of note names so we need to look at how to translate actual written music.

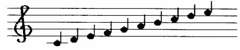
When music is written on paper the notes are shown as large dots with vertical tails. These note symbols are drawn on, or between, a series of horizontal lines called a *stave* and will appear as shown in Fig. 10.9. The position of the note on the stave indicates its pitch so that the higher the note symbol is drawn the higher will be its pitch. Successive natural notes in the scale are drawn on and between the stave lines.

Note	C	D	E	F	G	A	B	C	D	E
Pitch No.	0	2	4	5	7	9	11	12	14	16

Fig. 10.9 The treble music stave and note pitch values

There are in fact two musical staves and the one shown in Fig. 10.9 is known as the treble stave. The symbol at the left end of the stave is called the *treble clef* and simply identifies this set of lines as the treble stave. The treble stave shows notes above Middle C which is the note that sits on its own short line just below the treble stave.

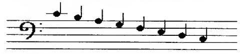
For the notes below Middle C there is a second stave which is called the *bass stave* and is shown in Fig. 10.10. Once again a special symbol at the left side called the *bass clef* identifies this stave. In the bass stave the Middle C note appears on a short line of its own above the bass stave.

Note	C	B	A	G	F	E	D	C
Pitch	0	-1	-3	-5	-7	-8	-10	-12

Fig. 10.10 The bass stave of music

Figures 10.9 and 10.10 also show the relationship between the notes as drawn on music staves to the pitch numbers that we need to use in the BEEP command on the Spectrum.

So far, however, we have only used the natural notes which correspond to the white keys on a piano. Most pieces of music will also use the black or sharp notes as well so we need to be able to recognise them in written music and tell the Spectrum to play them.

Sharp notes are shown in music by placing a crosshatch sign alongside the note as shown in Fig. 10.11. Sometimes a particular note is required to be sharp throughout the tune and this may be shown by placing a crosshatch sign at the start of each line of music at the position normally occupied by the note. This is also shown in Fig. 10.11. When the sharp symbol is placed at the start of the music stave then all notes on that line of the stave are made sharp instead of natural.

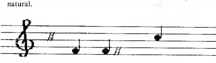



Fig. 10.11 The sharp notes

Telling the computer to play a sharp note is easy because we just add one to the pitch number for the basic note. The sharp note (C#) for Middle C will have the pitch value 1 since Middle C has the value 0. For a flat note we would simply subtract one from the basic note pitch number.

Now we can produce a simple tune by running the program listed in Fig. 10.12 which you may recognise. Although the notes have the

```
100 REM Tune playing program
110 FOR n=1 TO 26
120 READ p
130 BEEP .25,p
140 NEXT n
150 DATA 11,14,11,11,14,11,12
160 DATA 14,12,9,11,12,11,7
170 DATA 11,14,11,11,14,11
180 DATA 12,14,12,9,11,7
```

Fig. 10.12 A simple tune playing program right pitch values, the tune doesn't sound quite right. This is because all of the notes have the same length whereas in actual music the notes have varying duration to provide a rhythm to the tune.

Musical timing

Apart from the pitch, music also makes use of variable duration of the notes and this is organised on a binary system. The basic note length is called a *crotchet* and corresponds to a duration of about ¼ a second. The crotchet is shown as a black filled circle with a tail. Shorter notes are the *quaver* (1/2), *semiquaver* (1/4) and *demisemiquaver* (1/8) which are drawn like crotchets but have one, two or three ticks on the tail respectively. A longer note is the *minim* which is twice the length of a crotchet and is shown like a crotchet but with the circle not filled in. Finally there is the *semibreve* which is four times as long as a crotchet and is shown with no tail. These note symbols are shown in Fig. 10.13.

In the Spectrum we have to control note length by altering the duration of the BEEP command. A crotchet corresponds to a d value of 0.25 and the other note lengths are in proportion as shown in Fig. 10.13. This would normally involve the use of a whole series

Note	ratio	name	length
	1/8	Demisemiquaver	1
	1/4	Semiquaver	2
	1/2	Quaver	4
	1	Crotchet	8
	2	Minim	16
	4	Semibreve	32

Fig. 10.13 The various note lengths.

of fractional numbers for d. One solution for this might be to set up a basic value for duration of 1/32 second by using the variable c. This gives the note length for a demisemiquaver which is the shortest note. Now we can set up a note length number 1 which is proportional to note length so that a quaver has a 1 value of 4 and a minim has 1 value of 16. In the BEEP statement the duration is obtained by multiplying c by the required note length 1.

Now we can apply these new duration values to our tune generating program as shown in the program listing of Fig. 10.14. In this program two data arrays are used, one giving the pitch p and the other the length 1. The values of p and 1 are then used in turn in the BEEP statement to play the tune which now begins to sound much better.

```
100 REM Tune playing program
110 REM with varying note length
120 LET c=1/32
130 FOR n=1 TO 26
140 READ 1,p
150 BEEP c*1,p
160 NEXT n
170 DATA 8,11,4,14,8,11,8,11
180 DATA 4,14,8,11,4,12,4,14
190 DATA 4,12,8,9,4,11,4,12
200 DATA 4,11,8,7,8,11,4,14
210 DATA 8,11,8,11,4,14,8,11
220 DATA 4,12,4,14,4,12,8,9
230 DATA 4,11,8,7
```

Fig. 10.14. Improved version of the tune program.

The binary series of notes do not however satisfy all of the needs of musicians, so sometimes in written music you may also come across notes with a dot alongside the note symbol as shown in Fig. 10.15. These notes have the duration increased by half. Thus a dotted crotchet would have an effective duration multiplier value of 12 instead of the normal 8 and a dotted quaver would have a 1 value of 6 instead of the normal 4.

Another feature of music, apart from the duration of the notes themselves, are the pauses between notes which are known as *rests*. Special symbols are used to indicate these on the music stave and they are as shown in Fig. 10.16. These pauses can be introduced into

Note	ratio	length
	1/8	1.5
	1/4	3
	1/2	6
	1	12
	2	24
	4	48

Fig. 10.15 The dotted notes and their duration

the Spectrum music by using the PAUSE command of the Spectrum but some method will be needed to tell the Spectrum that a PAUSE is required rather than a note. One possibility might be to make the note length negative when a pause is required, then a simple check on the duration number will indicate which type of instruction is required. The alternative is to have a third data array for rests. This would have the value 0 when a note is to be played and a duration number when a pause is required. Note that since the PAUSE command works in 1/50 second units the lengths may not be exactly correct but should be near enough to produce acceptable music.

Symbol	Ratio	PAUSE value
	1/16	1
	1/8	2
	1/4	4
	1/2	8
	1	16

Fig. 10.16 The pause or rest symbols in music

Changing the tempo

Nomally music would be played with the length of a crotchet set at about 0 25 second. If this duration is made less then the tune will be played faster whilst if the crotchet length is increased the tune is played slower

Try running the program listed in Fig. 10 17 Here the basic scale from middle C up through one octave is played at increasing tempo until eventually all of the notes run into one another to produce a sound that might be useful in an arcade game You might also try the same idea but with the notes running down the scale

```
100 REM Scale with tempo change
110 FOR t=0.25 TO 0 STEP -.02
120 RESTORE
130 FOR n=1 TO 8
140 READ p
150 BEEP t,p
160 NEXT n
170 PAUSE 25
180 NEXT t
190 DATA 0,2,4,5,7,9,11,12
```

Fig. 10 17 Applying tempo change to a scale to get sound effects

A point to note here is that as the time gets shorter some notes get quieter and eventually disappear. This is because there just isn't enough time to generate one cycle of the required note.

Play the Spectrum

So far we have produced music by setting up the stream of notes and then playing them one after another with BEEP commands There is a rather more attractive possibility with the Spectrum and that is to turn the computer into a playable instrument.

We can use the Spectrum keyboard to act in the same way as the keyboard of a piano by allocating notes to a selection of keys. When one of these keys is pressed the corresponding note will be played by the spectrum

The first step is to choose the keys to be used for producing musical notes. The obvious choice is to try to get a key layout that is similar to that of say a piano. With this in mind the top two rows of letter keys were chosen. The middle row of letter keys running from A to L is used for the natural, or white, notes starting with key A which produces the Middle C note. Some of the keys in the row above, which runs from Q to P, are used for the sharp or black notes.

To detect a key press we can use INKEY$ which produces a string variable a$. If no key is pressed the string a$ will be blank and this is checked in line 180. A blank string simply makes the computer repeat the INKEY$ operation. When a key is detected the next step is to choose and set up the note data for the BEEP command.

The simplest technique for setting up the notes is to produce an array p with a value for each of the letter keys. In fact since we know that Z will not be used there are only 25 slots in the p data array. Into each slot is placed a pitch number. For those keys which are to be used for notes the pitch of the note corresponding to the key is used.

```
100 REM Play the Spectrum
110 DIM p(25)
115 REM Set up pitch table
120 FOR n=1 TO 25
130 READ p(n)
140 NEXT n
150 DATA 0,64,64,4,3,5,7,9,64,11,12,14,64
160 DATA 64,13,64,64,64,2,6,10,64,1,64,8
170 GO SUB 500
175 REM Note playing loop
180 LET a$=INKEY$: IF a$="" THEN  GO TO 180
190 BEEP .25,p(CODE a$-96): GO TO 180
200 STOP
490 REM
495 REM Display drawing subroutine
500 LET b$=" "+CHR$ 133+CHR$ 138
510 FOR r=1 TO 4
520 PRINT AT 3+r,3;b$;
530 FOR k=1 TO 7
540 IF k=2 OR k=6 THEN  PRINT "   ";: NEXT k
550 PRINT b$;: NEXT k
560 NEXT r
570 PRINT AT 2,4;"w  e     t  y  u     o";
580 PRINT AT 13,3;"a  s  d  f  g  h  j  k  l"
590 FOR k=0 TO 8
600 PLOT 16+k*24,80
610 DRAW 24,0
620 DRAW 0,64
630 DRAW -24,0
640 DRAW 0,-64
650 NEXT k
660 PRINT AT 17,1;"Play now";
670 RETURN
```

Fig 10.18 The Spectrum as an instrument program

For all other keys a value of 64 is used which produces a note too high to be heard.

The simplest technique for setting up the notes is to produce an array p with a value for each of the letter keys. In fact since we know that Z will not be used there are only 25 slots in the p data array. Into each slot is placed a pitch number. For those keys which are to be used for notes the pitch of the note corresponding to the key is used. For all other keys a value of 64 is used which produces a note too high to be heard.

The key codes for the lower-case letters A to Y start with a value of 97 for A and increase as we work through the alphabet. In the BEEP command the code for the key that has been pressed is found by using CODE and then 96 is subtracted to give a number from 1 to 25 according to which letter key was pressed. This number selects the item from the p array and therefore selects the required pitch value for the BEEP command. A basic note length of 0.25 was chosen for the BEEP command but this could be changed if desired to give a different feel to the keyboard.

Figure 10.18 lists a program which turns the Spectrum into a keyboard instrument. Before the program goes into the play mode a diagram showing the key layout is drawn on the screen to help you find your way around the keyboard. The screen display is as shown in Fig. 10.19.

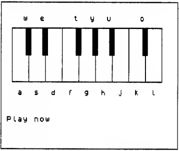


Fig. 10.19. Screen display for instrument program.

Other keys might also be used to provide functions such as shifting the octave that is being played. To do this you would simply add 12 to the pitch number to go up an octave or subtract 12 to go down an octave.

Index

LIGHT UP YOUR SPECTRUM!

This book shows you how to make the most of the Spectrum's graphics and sound capabilities, and you will be amazed by what you can achieve.

This book guides you through the basic principles of graphics on computers, and then covers the techniques of drawing, producing graphs and using colour. Later chapters deal with animation, including the techniques required for writing games programs and also three-dimensional displays. You are shown how to produce sound effects and music, and exploit the Spectrum's creative potential.

Many short, easily handled program listings are provided as well as several complete listings for you to try out and enjoy.

The Author
Steve Money is a well-known author of several books including *Microprocessor Data Book* published by Granada.

GRANADA PUBLISHING
Printed in Great Britain 0 246 12192 0

£6.95 net